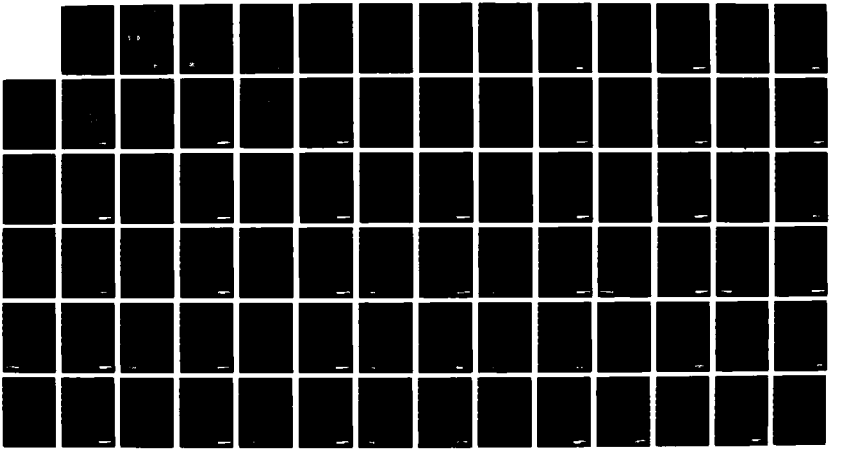
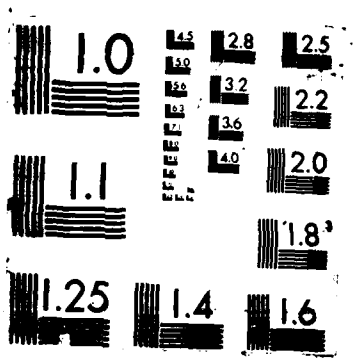
AD-A187 290 AN ADVANCED PROTOTYPING TOOL FOR HUMAN FACTORS DESIGN 1/1
(U) APPLICATIONS RESEARCH CORP DAYTON OH T V BROWN
MAR 87 ARC-TR-8782 DAAA15-86-C-0063

UNCLASSIFIED F/G 23/2 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3

ARC-TR-8782

AN ADVANCED PROTOTYPING TOOL FOR HUMAN FACTORS DESIGN

FINAL REPORT

CONTRACT: DAAA15-86-C-0063

BY

THOMAS V. BROWN

MARCH 1987

SUBMITTED TO

DR. BENJAMIN E. CUMMINGS
HUMAN ENGINEERING LABORATORY
ATTN: SLCHE-SP
ABERDEEN PROVING GROUND, MARYLAND 21005-5001

Applications RESEARCH Corporation

4031 Colonel Glenn Highway, Dayton, Ohio 45431 (513) 429-0200

ARC-TR-8782

AN ADVANCED PROTOTYPING TOOL FOR HUMAN FACTORS DESIGN

FINAL REPORT

CONTRACT: DAAA15-86-C-0063

BY

THOMAS V. BROWN

MARCH 1987

SUBMITTED TO

DR. BENJAMIN E. CUMMINGS
HUMAN ENGINEERING LABORATORY
ATTN: SLCHE-SP
ABERDEEN PROVING GROUND, MARYLAND 21005-5001

APPROVED FOR RELEASE

PAUL D. ZIDEK
VICE PRESIDENT

Applications RESEARCH Corporation

4031 Colonel Glenn Highway, Dayton, Ohio 45431 (513) 429-0200

EXECUTIVE SUMMARY

Modern equipment has placed new demands on human factors because of more complex machine/computer functions and increased speed demands. The human operator must interface with mechanical systems, computer software, artificial feedback effects, and other humans. There is a need for a formal, explicitly defined methodology that can be used to model all of these interfaces within one context.

This report describes a methodology that satisfies this need. This is a decompositional methodology that is general enough to include all system interactions, including machine effects and psychological characteristics of a human operator. The methodology is also rigorous enough to identify gaps and inadequacies to the user.

Its foundation is the USE.IT methodology which was designed to decompose software tasks.

Although the USE.IT methodology was not created under this effort, its application and demonstration to human factors problems did originate in this effort.

An interactive intercom system, a body motion modeling task involving walking and lifting, and a human operator interacting with a radio to keep it tuned are included as examples.

In each example, the methodology focuses, at each level of the decomposition, on the list inputs and outputs to each process at that level. The human factorist knows, at each level of the decomposition, exactly what has to be produced and what is available to produce it. The decomposition proceeds according to several basic ways of breaking down a complex task into several simpler tasks.

The examples demonstrate that the methodology is applicable to human factors interface problems. The methodology has broad application to different kinds of modeling systems within human factors. Perhaps most important, it was shown in the examples how human factors modeling with this methodology can incorporate additions of psychological phenomena, e.g., errors by the human operator, misunderstandings, delays in response.

Several recommendations are made. A library of basic human factors functions, already decomposed into a set of more basic functions would extend the usefulness of the methodology. A library of functions will allow human factors modeling data to accumulate modeling as it is done. In effect, this will allow reusability to be achieved for model components with considerable savings.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

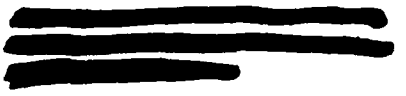
1.0 THE NEED FOR A FORMAL HUMAN FACTORS DESIGN TOOL

The human-factors discipline has grown in complexity commensurate with the growing complexity of man-machine interfaces. Man-machine interfaces consist not only of displays and mechanical devices, but computer hardware, software, processes, and similar factors. Time and capability demands have also increased, and recently, artificial intelligence has been added to the interface system. These demands require a design tool which integrates all aspects of the man-machine system within one methodology.

1.1 A FORMAL SYSTEM

A formal design tool is required for understanding and integrating elements of complex systems. A human designer can only comprehend a limited number of alternatives and contingencies. Using formal methods, a designer can cope with a large number of variables and their complexities in an orderly and thorough way. Formal tools are well-defined ones which require users always to work within a defined set of rules.

A lack of formal definitions is a typical cause of problems related to block diagrams. Often, when an engineer lays out a system's block diagram, part of the diagram employs new ad hoc symbols to illustrate some special aspect of the system. Figure 1 shows examples of these. If the block diagrams do not have a well-defined set of rules governing their construction, users invent new symbols, and these are usually ambiguous to subsequent users. Obviously, confusion has been added since users must now comprehend the new symbols before the basic block diagram can be understood.



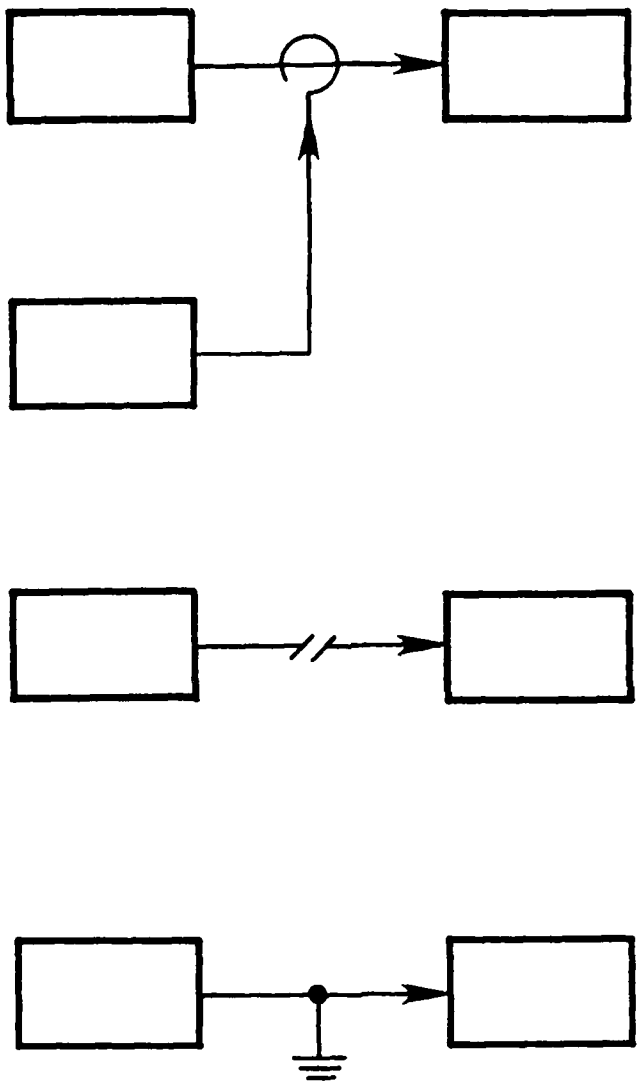


Figure 1 Examples of Unclear, AD HOC Block Diagram Notations

A formal tool generally allows some form of block manipulation; that is, blocks can be transformed with confidence within the context of the system description. Figure 2 illustrates a very simple form of block diagram computation, which is valid for linear systems. The formal system rules support the computation.

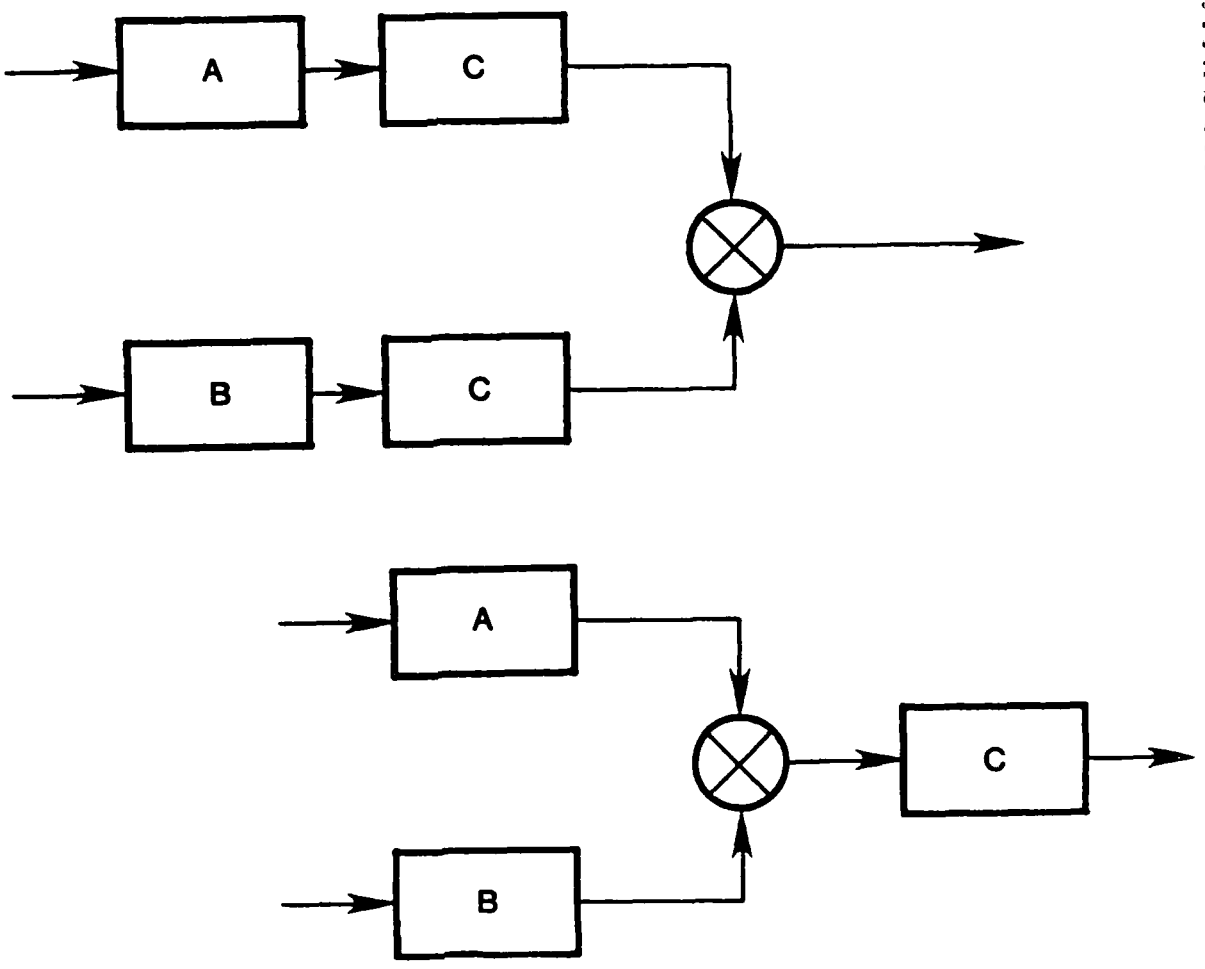


Figure 2 A Way of Computing with a Block Diagram -- Two Versions Are Equivalent for Linear Systems

A computational facility allows a designer to be considerably more thorough. For example, if a system is described in terms of the formal, mathematical rules of a Petri Network, there is a mathematical way to prove that the system will never reach a deadlock condition. This is a powerful conclusion which could never be reached for even moderately complex systems without formal computational methods.

Because of the increased complexity of equipment and man-machine interfaces, the tools used by human factorists to evaluate and develop complex systems have been stressed to the point where new tools need to be developed. To meet this expanding complex situation, new tools must possess specified characteristics. Five of these characteristics are considered below.

First, the tools themselves should be as automated as possible. Computer software is well suited to track many detailed facets of a problem. A good automated tool reduces costs by accelerating design and analysis, supplanting work which was done manually. Automated tools must be thorough and deal with all the tedious details that human analysts are prone to overlook. Automation can also facilitate the testing of new system concepts and modifications.

Second, tools should provide a strong basis for organizing within a context; when a particular approach to an interface problem is taken, analysts accumulate knowledge and experience. An organizing tool must be sufficiently general to permit many different points of view to be applied when the tool is used. If well organized, analysis can be easily documented, enabling design recommendations to be supported by studies and requirements.

Third, a facility for prototyping would be extremely useful. Prototyping allows quick conceptual evaluations of interface approaches to be made before committing large investments of time and money. Demonstrations of "what if"

modifications could be accomplished using a prototyping tool; modifications under consideration could be prototyped before incorporation into a system.

Fourth, a good human-factor analytical tool should support some form of simulation. Ideally, the tool should produce source code capable of simulating a human operator and incorporation into larger equipment simulations.

Fifth, if the analytical tool could produce specifications and documentation which fulfill applicable MIL-STDs, additional savings result.

2.0 USE.IT AS A HUMAN FACTORS DESIGN TOOL

2.1 DESCRIPTION OF USE.IT

This report addresses an opportunity to apply a software design tool to human-factor problems. This tool--the USE.IT system--has all the characteristics needed to evaluate man-machine interface for systems undergoing concept evaluation and development.

USE.IT was created as a software design tool by Higher Order Software, Inc. Software requirements are entered into a computer, checked for consistency and compatibility, and then used to generate executable source code directly. USE.IT has been written, programmed for use on a computer, documented, tested, and is available now. Its applications have extended beyond its software design intent but not yet into human factors. Phase I of this proposal takes advantage of the powerful capabilities of USE.IT, and demonstrates how to apply them to analyze complex man-machine interfaces.

Figure 3 illustrates the USE.IT system as used to generate software. The AXES component is the graphical input section in which a system specification is entered into memory. The ANALYZER checks the system description for consistency to assure that all interfaces are satisfied, i.e., two sides match in terms of what each expects from the other. The resources allocation tool (RAT) section

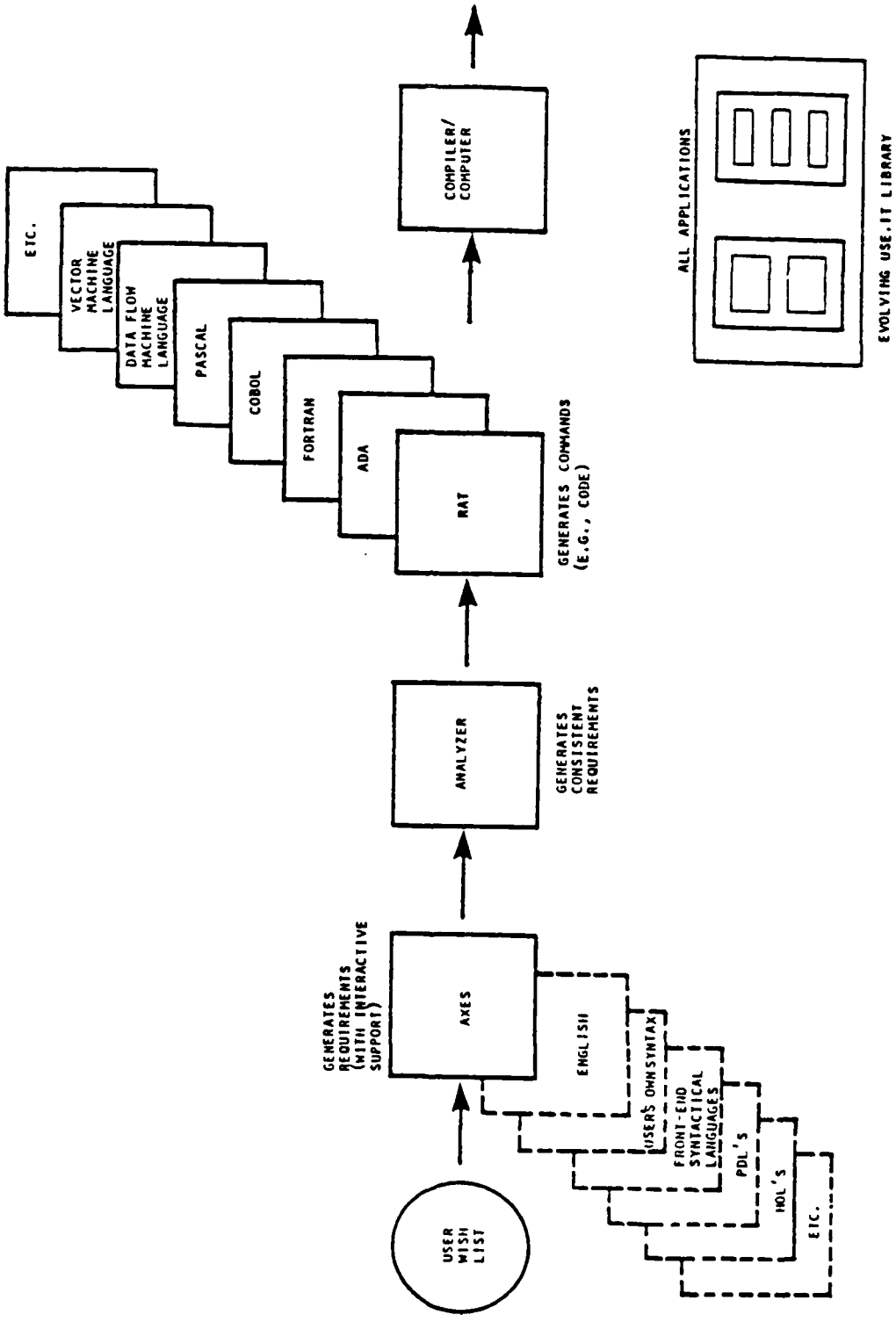


Figure 3 USE.IT Methodology

generates executable source code; there are RAT modules available for different computer languages, including BASIC and FORTRAN.

The strength of the USE.IT approach is that complex systems are decomposed into simple components, down to a primitive level. The hierarchical decomposition follows a set of forms, illustrated in Figure 4, which will assure that the specification is completely checked for consistencies and that software code generated by the RAT is correct.

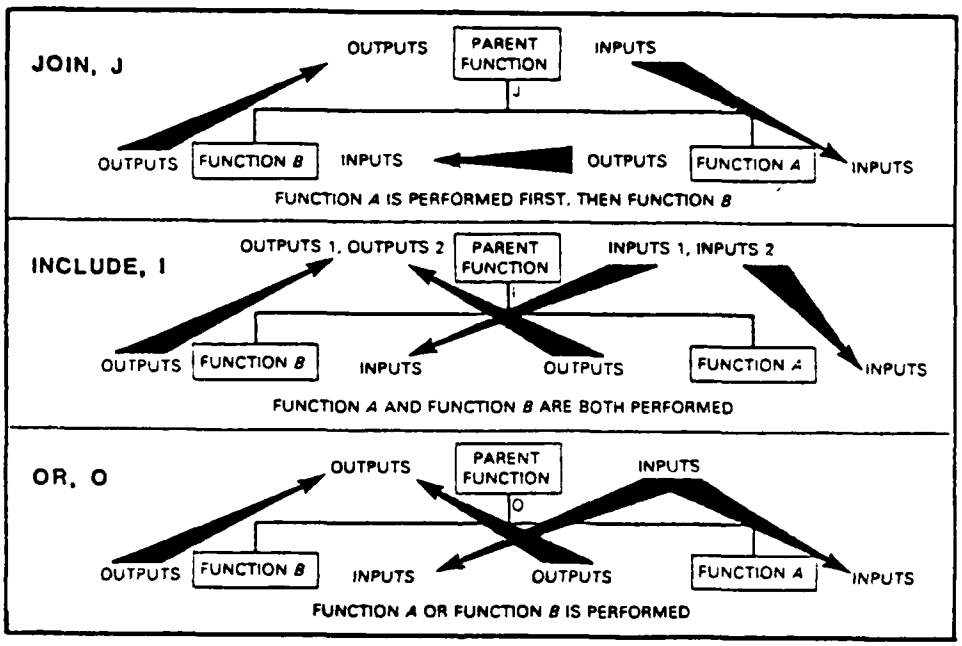


Figure 4 Basic Decomposition Forms of USE.IT

Even though there are only three basic decomposition forms, USE.IT can combine these forms to create a variety of structures which can represent any form of complex conversions, interactions, or system behavior. USE.IT supports the five desirable characteristics for human-factor analysis outlined above.

First, USE.IT automates all operations beyond the initial specification or system description. The input is entered at a graphic terminal following the USE.IT format. This information is then stored for checking, prototyping, and generating source code. USE.IT is currently available, thereby obviating development costs. Only its application needs to be demonstrated. The USE.IT design is based on rigorous mathematical proofs; therefore, checking and code generation are achieved with the certainty of mathematical proofs. Since all input is through a graphics terminal and keyboard, no programmer is needed to access the full power of the software. All work involving USE.IT can be accomplished much faster than with conventional methods; USE.IT speed has been demonstrated for different applications, software and non-software.

Second, USE.IT has a built-in library of system components so that library components can be instantly incorporated into a specification. The library allows a design to be organized and stored for later use. Collections of analytical components, each from a different human-factor viewpoint, can be stored in the library. USE.IT encourages and supports modularity. Documentation describing each library component can be carried internally as it is manipulated and ready to be printed/plotted when needed.

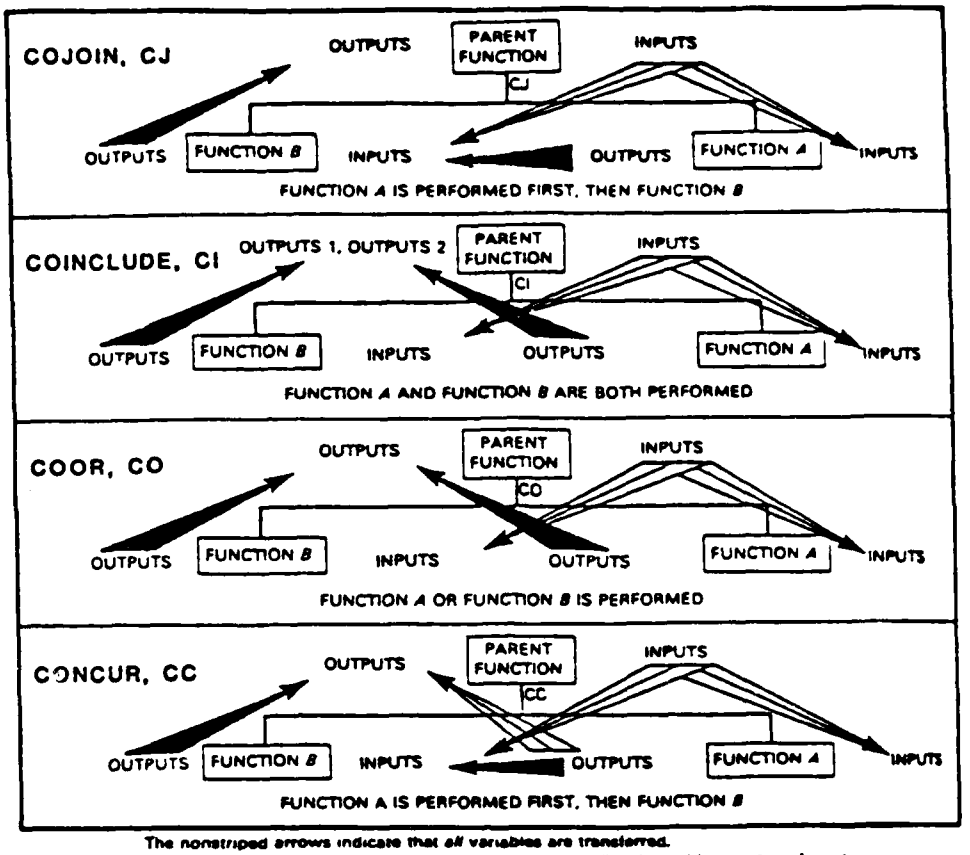
Third, USE.IT supports prototyping at any level. A prototype is automatically generated within the system specification by a simple command. This command generates software which displays the computed inputs to the prototype module, and asks the user to enter the outputs which they believe the module will generate. Program execution then continues automatically until the next prototype menu is shown on the CRT. The combination of prototyping and library modules allows quick demonstrations of conceptual systems, which can be built with existing library components or recently entered components. Once a system is designed,

it can be tested thoroughly for consistency both within modules and across interfaces. All errors are indicated. If a system is changed and new components added, the new design can also be thoroughly checked to assure that the changes improve and not degrade performance.

Fourth, the USE.IT facility for generating source code in more than one language (FORTRAN, Pascal, machine language) can be used to allow simulation source code to be written for incorporation into larger machine simulations. By including certain simulation modules within the USE.IT library, the automatically generated software can support simulation of the man-machine interactions. Of course, prototyping can be used within this executable code if desired.

Fifth, the USE.IT development process can create intermediate products which fulfill MIL-STD requirements, i.e., MIL-STD-490, Type A and B specifications.

USE.IT employs a decompositional process for each function or object, and has a set of inputs and a set of outputs. The overall system is represented by the highest-level function. Each parent function is divided into two offspring functions according to a provably correct format. This format, illustrated in Figure 2, assures that the entire system is correct and operates as designed. The three basic forms shown in the figure are building blocks of other functions. Figure 5 illustrates several derived functions which are built using combinations of basic decomposition forms. These derived forms and hundreds of others are kept in the USE.IT library to be called as needed for insertion into the system. There is a facility for including generic forms within the library so that a complex structure can be particularized by describing the functions of several components. For example, an object, such as an intercom system, can be described completely in terms of possible inputs and outputs. This object's structure can be kept in



The nonstriped arrows indicate that *all* variables are transferred.
The striped arrows indicate that *some but not necessarily all* variables are transferred.

Figure 5 Derived Decompositional Forms Commonly Used in USE.IT

the library and used as needed. Inputs and outputs to the intercom object would be quite general, i.e., mike button pressed, sound entering the microphone, and electrical power requirements.

Note that inputs and outputs can include physical quantities related to hardware and software values, human factors inputs and outputs, and any other expressible term. Inputs and outputs can also include sets of other values. USE.IT contains a data analysis interface such that when components of a data element are used, the data element itself must be provided.

The basic decomposition forms, JOIN, INCLUDE, and OR, are defined in terms of how parent function inputs are transformed into outputs. OR allows for contingency conditions based on the value of some Boolean variable.

JOIN is a two-stage function in which inputs are transformed into intermediate variables, subsequently to be transformed into outputs. INCLUDE breaks up the set of inputs so that some are transformed into output values by each offspring component. If all functions can be built from these three functions, the correctness of the derived functions is assured.

Derived functions such as those shown in Figure 5 are built from basic forms, as are all functions which enter the USE.IT library. USE.IT software automatically checks for this consistency.

Forms can be called from the library for representation on a graphics terminal where the user combines the forms and enters the descriptions necessary to describe a human-factor situation. Programming skills are not prerequisites to employing USE.IT software. In non-software design applications of USE.IT, work has been accomplished many times faster than with manual methods. For example, in one application, a building was designed to a customer's specifications by one person employing USE.IT in only 11 days, whereas normally it would have required at least 1000 person days. The person who designed the building was not a programmer and had little experience with USE.IT. Non-soft-ware applications of USE.IT are not new, but its application to man-machine interactions is new.

2.2 AN EXAMPLE OF AN APPLICATION OF USE.IT

To show how to apply USE.IT as an advanced human-factors tool, an example of another interface analysis technique will be shown for contrast. Limitations of this technique will be indicated. USE.IT will then be applied to the same example situation to show how to overcome these limitations.

Any interface tool or descriptive notation must be able to incorporate precedence relationships and contingencies among the events along a time line, i.e., the vertical axis of Figure 6. Contingencies are complex, but can be handled

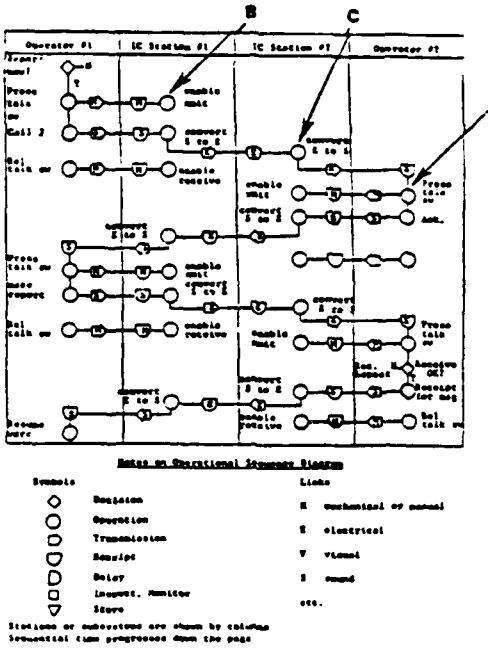


Figure 6 Example of Operational Sequencing Diagramming

by older methods such as Petri network analysis. However, as computationally rigorous as Petri nets are, their diagramming is visually complex, and its token-transfer procedure requires additional notation to adequately identify what is being transferred. Moreover, Petri network analysis is not yet automated as is the USE.IT system.

The interface analysis methodology used to illustrate an older technique is shown in Figure 6. Figure 6 from MIL-H-46855 illustrates a technique called Operational Sequence Diagramming (OSD). The system being analyzed is a two-station intercom with press-to-talk buttons. Time progression is shown down the page. Limitations of this approach are indicated by arrows A, B, and C. Arrow A indicates how human operator behavior and decision making are represented by a simple line from "receiving sound" to "pressing talk switch". Arrow B shows a sequence that terminates into nothingness at Station #1. Arrow C shows a conversion from "electrical" to "sound."

Arrow A shows that there is no provision for an operator to base his press-to-talk response on a particular sound, e.g., his call letters or some sound besides noise. Arrow B shows contingent effects -- the button must be pressed to transmit -- not included in the method. Arrow C shows a similar problem with receiving contingencies.

A USE.IT analysis of the two-station intercom demonstrated quickly that if operator #2 held his button down, the communication's process couldn't start. This is an unusual condition, but human-factor failures can easily occur under such conditions.

The proposed application of USE.IT as an advanced human factor tool is based on an object architecture point of view in which the situation is first conceptualized in terms of inputs and outputs, actual and potential, of the objects of the analysis. For the intercom system used in the OSD example above, there are two kinds of objects: an intercom station and an operator. There are two examples of each kind of object.

Figures 7 and 8 show how these objects are conceptualized in terms of inputs and outputs. Each object is mutually and independently described. Decoupling each object's description from other object descriptions has several advantages: (1) descriptions are conceptually easier; (2) a library of objects can be accumulated and thereby save resources; and (3) interface work can be delegated to several different groups, including both engineering and human factors groups.

Once control structures for all objects have been created, the master control structure, representing the communication of a report from one intercom station to another, is designed. In decomposing the master control structure, copies of the object control structures are called from the library as needed and substituted into the master control structure. The USE.IT program would check interfaces to assure that what feeds out of one structure matches the expected input to the other structures.

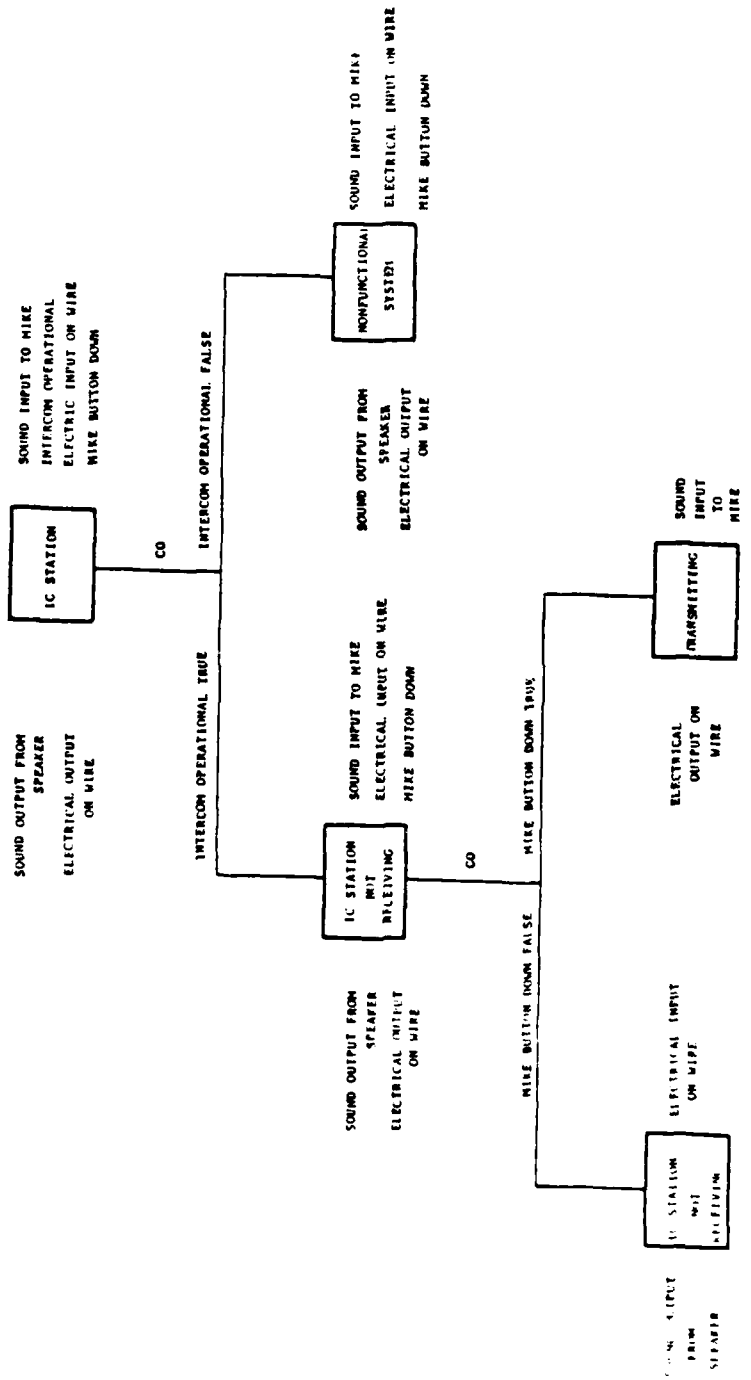


Figure 7 Intercom USE.IT Control Structure

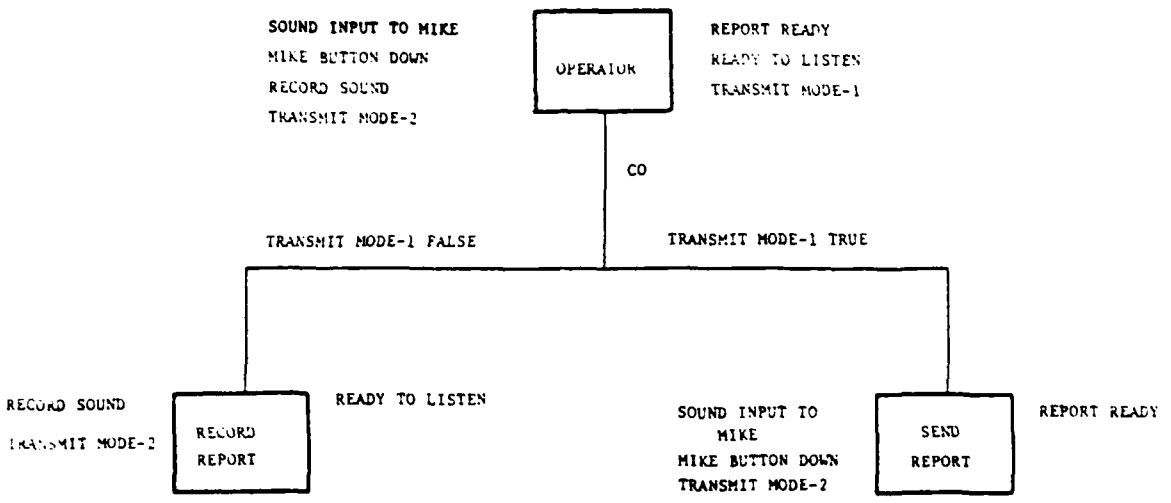


Figure 8 Intercom Operator USE.IT Control Structure

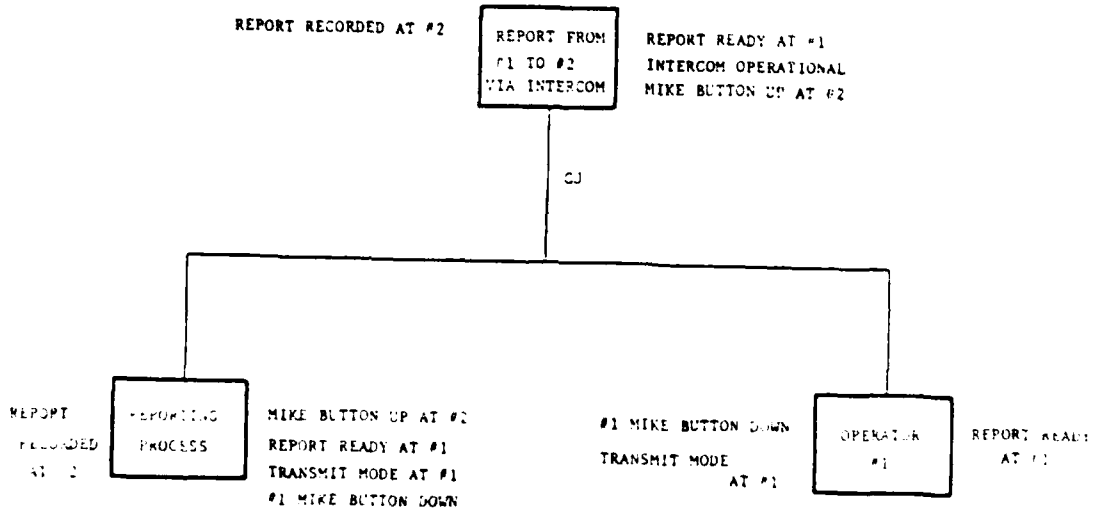
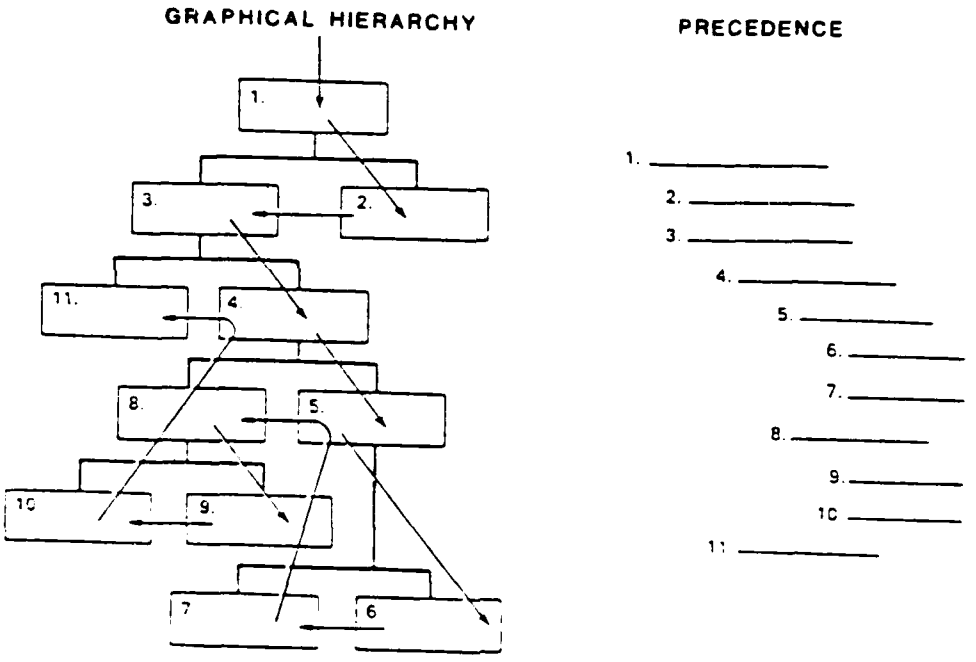


Figure 9 Master Control Structure Corresponding to OSD Example of Two Intercom Systems

Figure 9 shows how the master control structure would begin for the two-intercom example. Each block would be recursively decomposed until primitive, i.e., simple arithmetical, or library, structures were reached to terminate the branches of the decompositional tree.

As part of the USE.IT process, this scanning order of blocks within a control structure is determined by the simple scanning procedure shown in Figure 10. If executable code is generated, precedence is the order in which the code is executed.



The tree is represented in text using the following sequencing rules.

- Start at the top.
- If a node has offsprings, list the right offspring first.
- When the descendants beneath the right offspring are listed, list the left offspring

Figure 10 Determination of Precedence Order from Tree Structure

[redacted]

3.0 EXAMPLES OF HOW USE.IT CAN BE APPLIED TO HUMAN FACTORS PROBLEMS

The use of any decompositional process is virtually always an iterative one. As the content of one level is refined, there are implications for other levels. Sometimes an analysis at a very low level requires changes all the way up to the highest levels. USE.IT is no exception to this. In doing these examples, especially as a first application, several approaches to each level were almost always necessary.

Another dimension of the iteration was the data typing. Data typing is a very important and powerful aspect of USE.IT. As the model of a process evolves, the data types associated with the inputs and outputs for each process also are evolving. The final design requires complete type consistency.

The data element type determines which primitive processes can use that element as an input or output. The type also determines the possible set of values which that element can have.

For these applications, the Boolean type is used very frequently, always being limited to "TRUE" or "FALSE" values. The alphanumeric type consists of a string of characters. Other types can be layered, i.e., built on combinations of other data types. For example, an array of real numbers is a layered type, based on the real number type. More complex layering can include structures that contain different types. For example, a type defined as "orientation" might include three real values corresponding to Euler angles, a real value corresponding to time, a natural number denoting the version, and an alphanumeric type with the title of the orientation as a data element. Layered typing can represent a repository of knowledge about an area like human factors. Part of the expertise for any technical area is knowing what elements should be grouped together, e.g. what data typing is appropriate.

In these examples, a data dictionary will ultimately define the types for each input/output element. In cases which do not involve primitive processes, typing is not as crucial except for prototyping purposes. With more applications of USE.IT within human factors situations, a common set of layered data types will emerge, especially for particular applications.

An example of how typing is important involves the data type message being sent through a system. The message can be alphanumeric or oral. Its alphanumeric form can be alpha-numeric-typed, alphanumeric-handwritten, or alpha-numeric-electrical impulses. The oral forms can be oral-spoken or oral-spelled-out. This kind of distinction in typing is necessary and useful, for example, in processes shown in Figure 11. Note how psychological processes, such as reading, misunderstanding spoken words, short term memory, etc., can be included as processes under the blocks shown in Figure 11. Thus, this approach provides a means for including psychological modeling within a formal human-factors methodology.

Message_2 (Alpha-numeric electrical impulse)	Send telegraphic message	Message_1 (Alpha-numeric - typed)
Word_2 (Oral-spelled out)	Spell difficult words	Word_1 (Alphanumeric-typed)

Figure 11 Illustrations of Typing

3.1 EXAMPLE 1, EXTENSION OF TWO-STATION INTERCOM EXAMPLE

The two-station intercom example given in Section 2.2 used several levels of decomposition to illustrate the USE.IT methodology. In this section, the decomposition will be extended to more basic levels in which processes are more elemental. Figure 12 contains the control structure for the two-station intercom example. The data dictionary corresponding to Figure 12 is given in Figure 13.

Many blocks of Figure 12 are included to represent situational aspects which are easily neglected within a design process including equipment functions. For example, the INTERCOM_WORKING Boolean variable assumes that the INTERCOM_SYSTEM block will always yield correct outputs for whatever set of inputs are given. That is, the relationship between the inputs and outputs for each block is true for all possible combinations of input values. These relationships are the constants within the dynamics of the actual system.

Another aspect of the system which had to be included for the thoroughness demanded by USE.IT is the re-trying to establish communications only a finite number of times. After five trys, a failure of equipment is reported.

Since looping is involved, a recursive situation exists. USE.IT provides for looping so that it may be incorporated in a rigorous way into any system. Appendix A describes how to do this. Other control structures are possible with USE.IT.

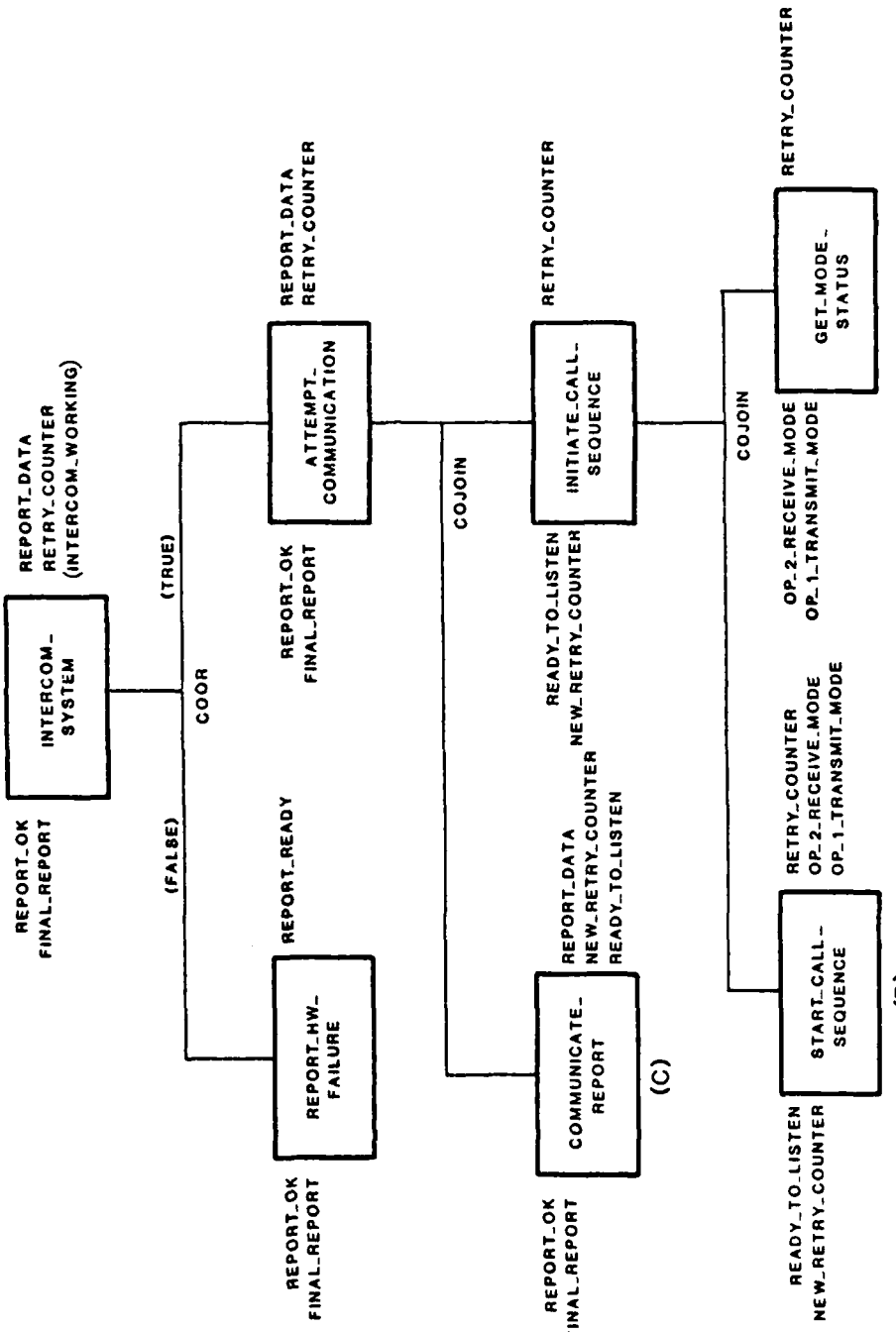


Figure 12A Control Structure for Example 1

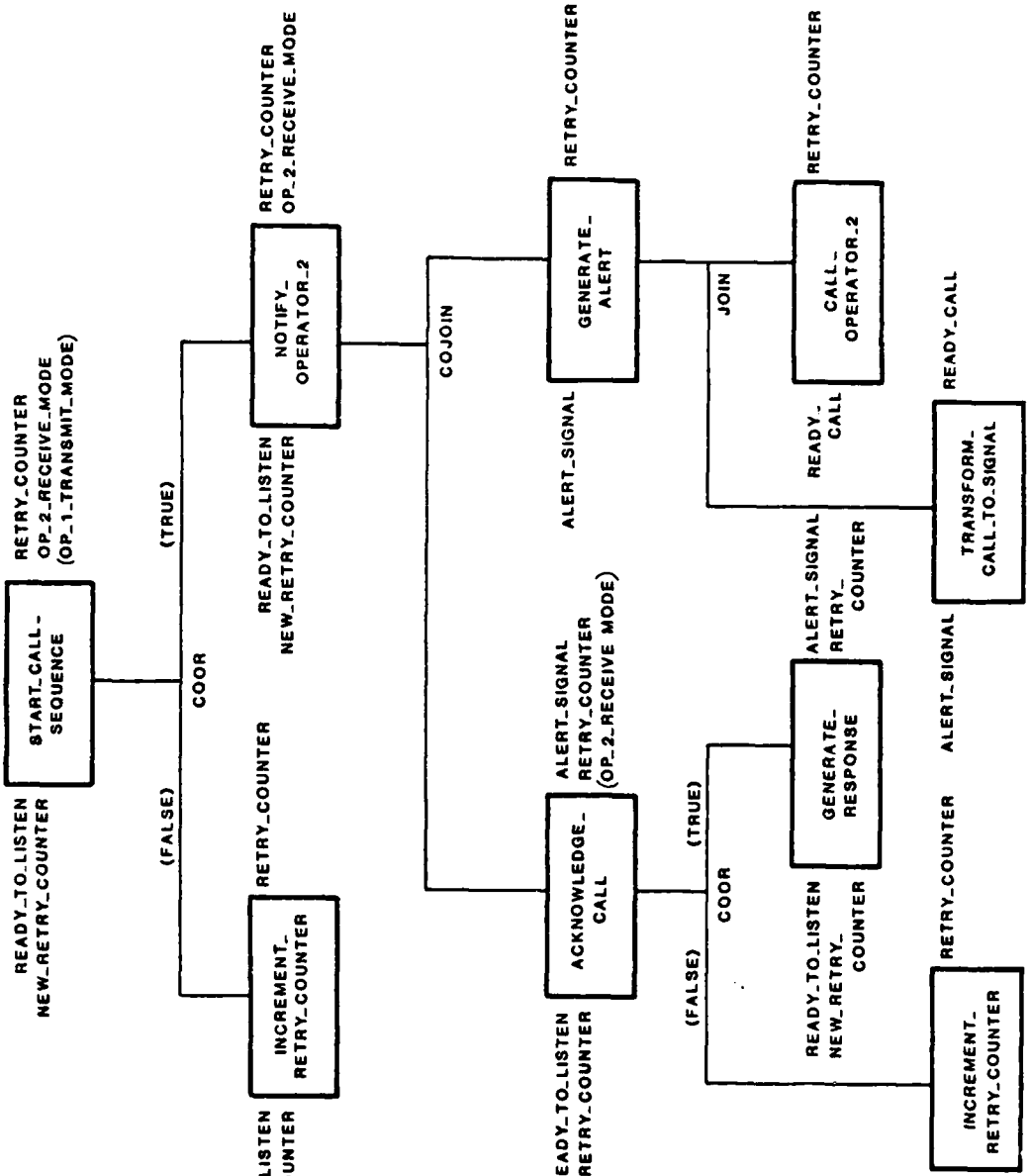


Figure 12B Control Structure for Example 1

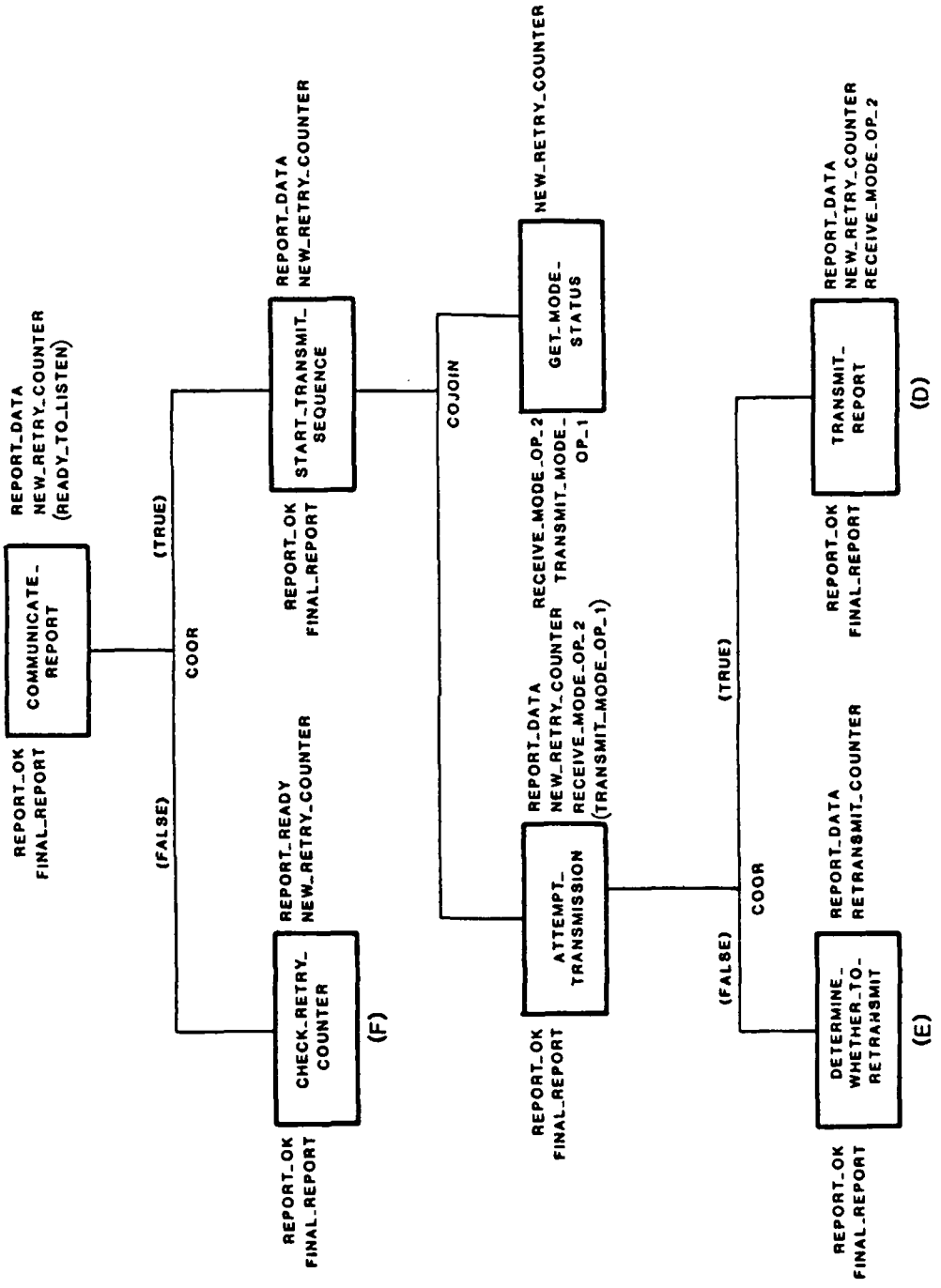


Figure 12C Control Structure for Example 1

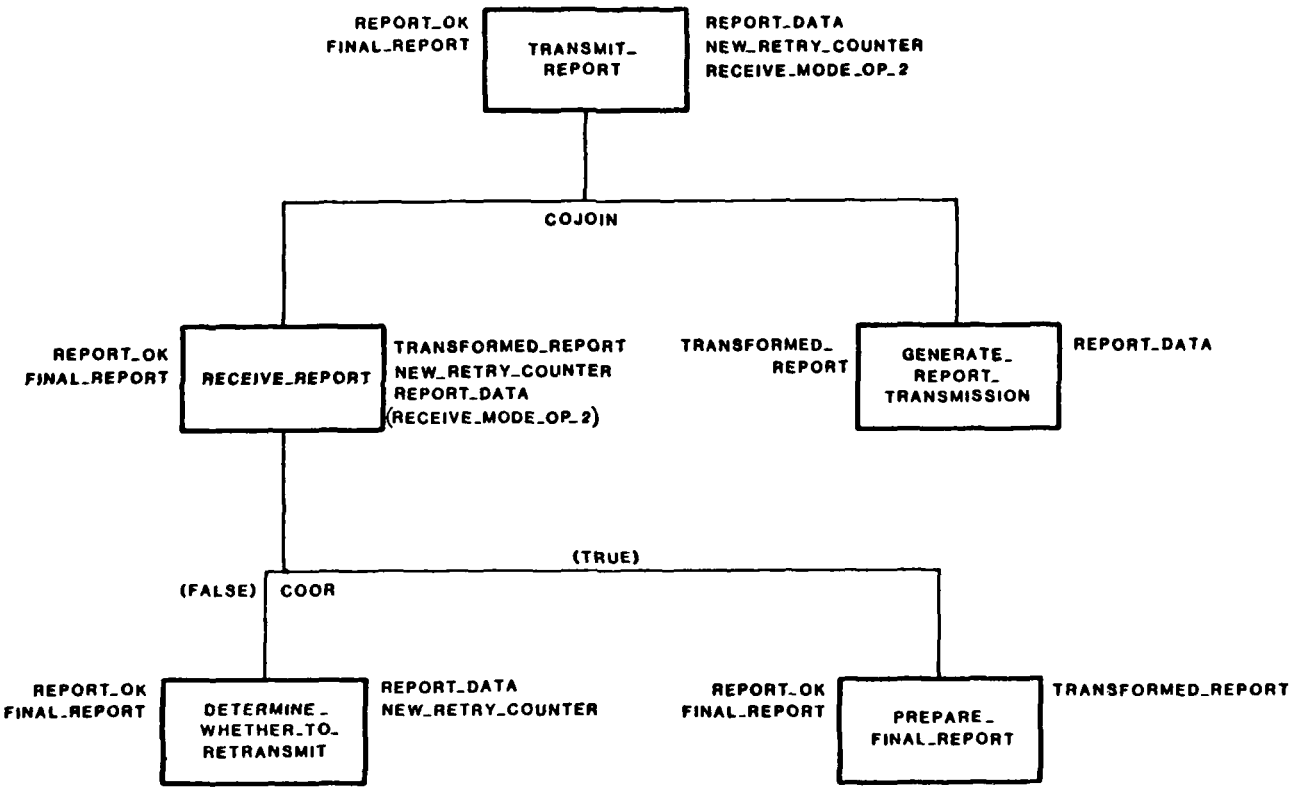


Figure 12D Control Structure for Example 1

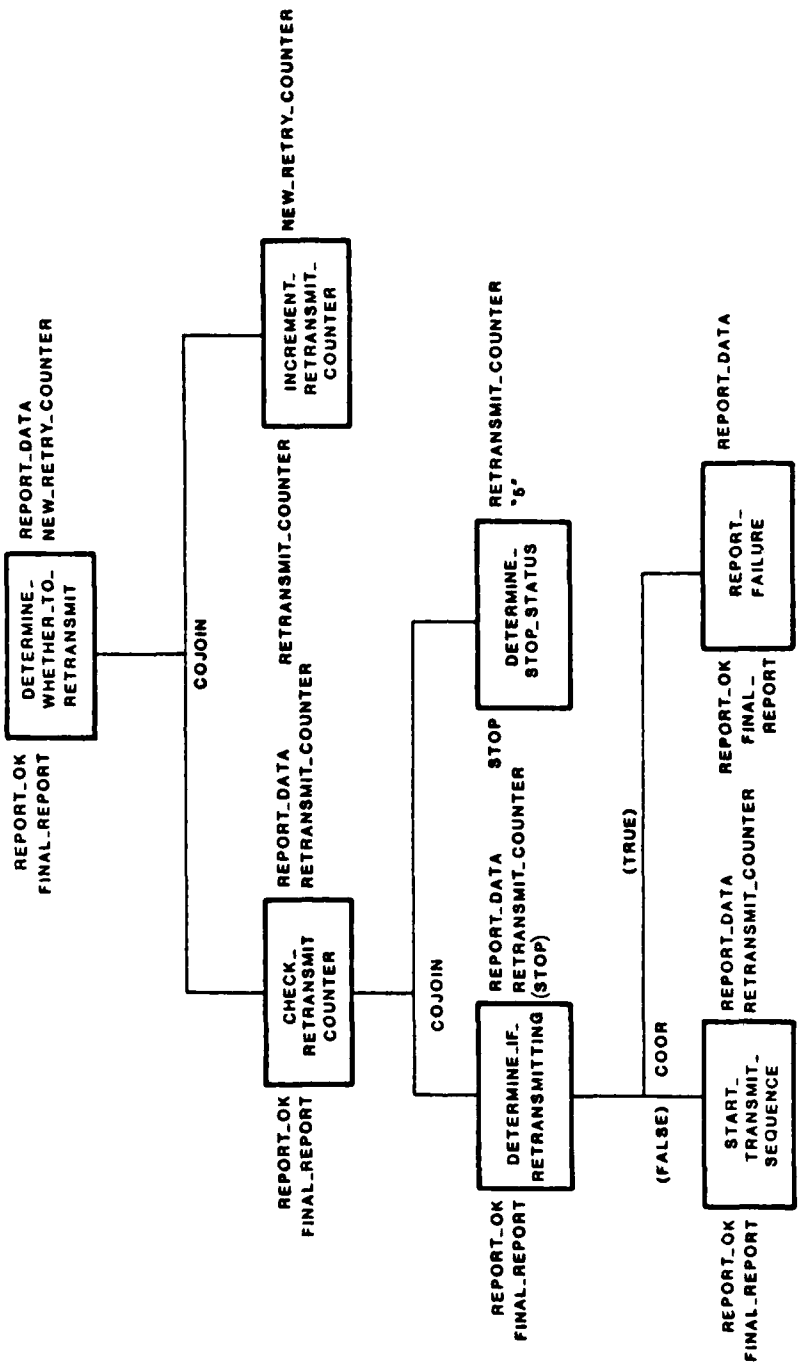


Figure 12E Control Structure for Example 1

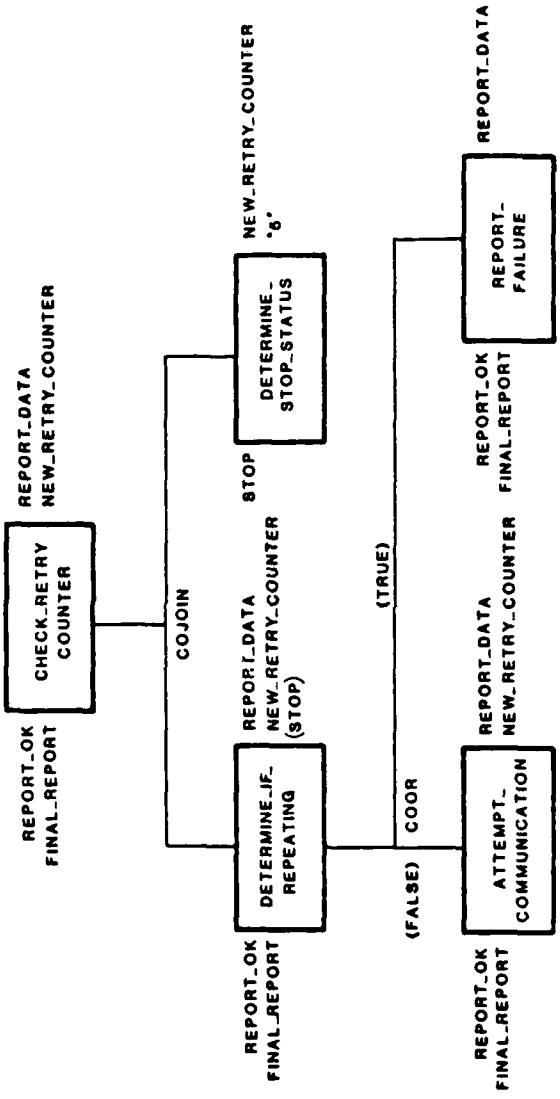


Figure 12F Control Structure for Example 1

DATA NAME	TYPE
ALERT_SIGNAL	ANY
FINAL_REPORT	ALPHANUMERIC
INTERCOM_WORKING	BOOLEAN
NEW_RETRY_COUNTER	NATURAL
OP_1_TRANSMIT_MODE	BOOLEAN
OP_2_RECEIVE_MODE	BOOLEAN
READY_TO_LISTEN	BOOLEAN
RECEIVE_MODE_OP_2	BOOLEAN
REPORT_DATA	ANY
REPORT_OK	BOOLEAN
RETRANSMIT_COUNTER	NATURAL
RETRY_COUNTER	NATURAL
STOP	BOOLEAN
TRANSFORMED_REPORT	ALPHANUMERIC
TRANSMIT_MODE_OP_1	BOOLEAN

Figure 13 Data Dictionary for Example 1

The purpose of this intercom system is to communicate data from Station 1 to Station 2. Additional requirements are that Station 2 be checked for availability before the data are transmitted and that up to five attempts to complete the transmission be made before aborting the effort.

The portion involved with checking the availability of Station 2 will be elaborated to demonstrate the points mentioned above. Inputs to the START_CALL_SEQUENCE are RETRY COUNTER, which tracks the number of times this communication has been attempted; OP_2_RECEIVE_MODE, which is TRUE only if Station 2 is ready to receive data; and OP_1_TRANSMIT_MODE, which is TRUE only if the transmitter is ready to transmit.

The READY_TO_LISTEN output is TRUE only if Station 2 has responded to the call from Station 1. NEW_RETRY_COUNTER is the updated RETRY_COUNTER. The welldefined interfaces allow independent development of this portion of the system without the mismatching of data that frequently is discovered during integration testing. Should the design for this section change and require different input, it will be easy to isolate which functions will be affected by tracing up the tree on the control map.

START_CALL_SEQUENCE is decomposed into INCREMENT_RETRY_COUNTER if OP_1_TRANSMIT_MODE is FALSE and NOTIFY_OPERATOR_2 if OP_1 TRANSMIT_MODE is TRUE. Regardless of which function is selected, the output variables for both functions are identical to those of the parent function.

NOTIFY_OPERATOR_2 decomposes into GENERATE_ALERT and ACKNOWLEDGE_CALL. These functions are performed sequentially, with the output of GENERATE_ALERT forming a part of the input to ACKNOWLEDGE_CALL. Here the design technique needs to be flexible enough to allow the designer to experiment as to where the man-machine boundary should go and how complex a system is to be developed. GENERATE_ALERT can be as sophisticated as a voice-operated transmitter that automatically starts working upon sensing the voice of the STATION-1 operator and the ALERT_SIGNAL is received at Station 2. The Station 2 system (GENERATE RESPONSE) may automatically send the response (READY_TO_LISTEN) if it is available to receive the data. At the other extreme, the same GENERATE_ALERT function can be accomplished by the operator at Station 1 pressing a button which causes either a light to flash or a bell (ALERT_SIGNAL) to ring at Station 2. As long as the HOS system design has well defined interfaces and the design adheres to the specifications on the control map, the integration problems will be kept to a minimum.

Although this example represented requirements definition, the same methodology can be used to expand this design to satisfy a detailed design. Only one methodology needs to be learned to satisfy both design needs. If the system required code generation for simulation or prototyping, it could be automatically produced by the USE.IT software without the problem of documentation not representing the code.

3.2 EXAMPLE 2, WALKING TO A BOX, LIFTING IT UP, AND PLACING IT ON A SHELF

In this example, a man starts from a standing position to walk across the room. He stops in front of a box on the floor, bends down, grasps the box, and lifts it upward to a shelf over the original location of the box. He then turns around and walks back to the place where he started, and turns around to return to his starting position.

There are several problem areas in this example. Since the walking motion is geometrically complex, there might be a tendency to start immediately with the definition of several coordinate systems to describe the motion. If this approach were followed, the designer or user would be faced with the problem of carrying this geometric data along at all levels of the analysis. Instead, specific geometrical data values will be postponed as long as possible downward into the decomposition. In this way, the designer is burdened with a minimum of terms and detail.

Modern software engineering uses the delay-of-detail approach in software design. Premature specificity has always made software design excessively complex. The same principle applies to human-factors modeling and design.

A second problem area is in the specification of parameters, i.e., step length, step size, height of shelf. Following the comments above, these details will be ignored until needed. Parameters associated with the man will be grouped in a layered data type, while parameters associated with the room will be

grouped together. Layering data means using a single name as a reference to a group of data items. In this way, parameters for each object will be grouped together and only elaborated when finally necessary. By delaying the elaboration the designer can isolate the specific parts of the control structure that depend on each group of parameters. For example, walking over to shelf is not a function of shelf height.

Cases will certainly arise cases in which the specification of some low-level details of the control structure will require new data that have not been passed to the lower-level blocks. This is part of the iterative nature of hierarchical designs. Delaying the elaboration allows these additions of detail to be done in a more orderly and formal way only when necessary.

Figure 14 is the control structure for this example. The process of walking over to the box and walking back to the start position requires looping for the steps. Walking starts with a short step, followed by a larger step, with a termination of a short step at the destination. This is complex to be appropriate for the USE.IT equivalent of a subroutine. The "Walk to New Position" control structure is shown in Figure 16. Figure 15 shows the step pattern which was used for the control structure given in Figure 16. Control structures like this would be kept within a USE.IT library. The use of a pre-defined control structure from a library is denoted by an "OP" under the block which is to be replaced by the library control structure.

In the process of creating the control structure shown in Figure 16, several points became evident. In Figure 16A, it became necessary to include data related to the starting position of the man--MAN_START. In order to return to the start position after picking up the box, the location of the start must be retained. This became apparent only after the decomposition had proceeded downward several levels. A similar need arose to define the BOX_GOAL--where the box was to be placed--at the top level.

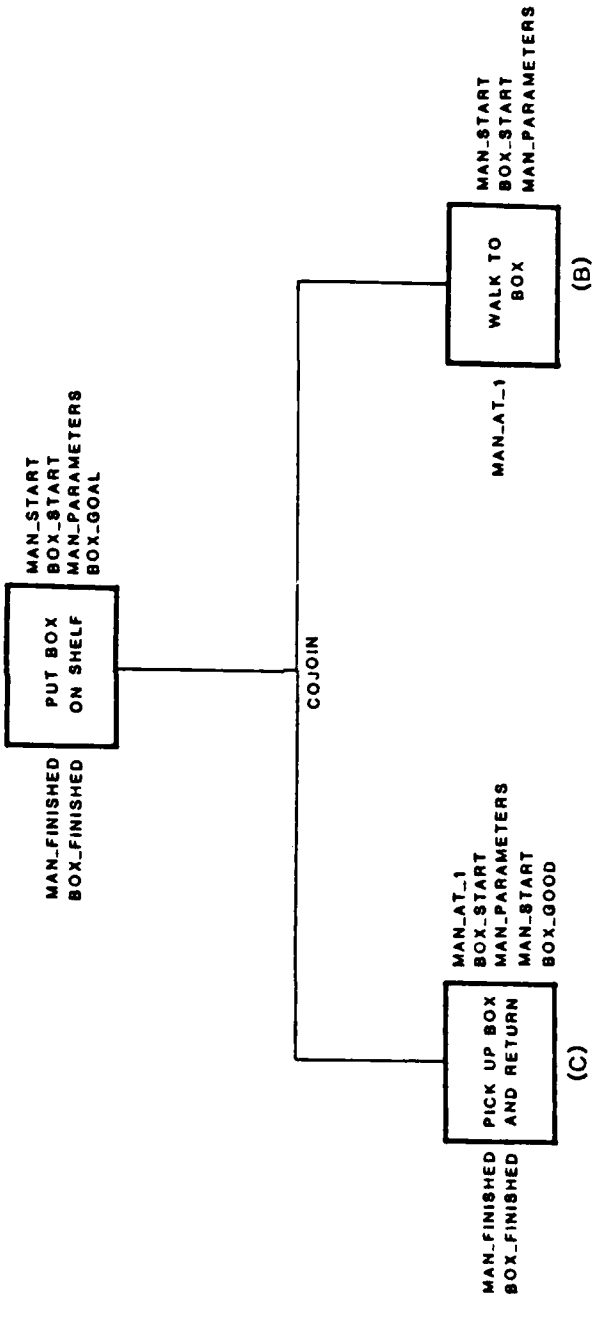


Figure 14A Control Structure for Example 2

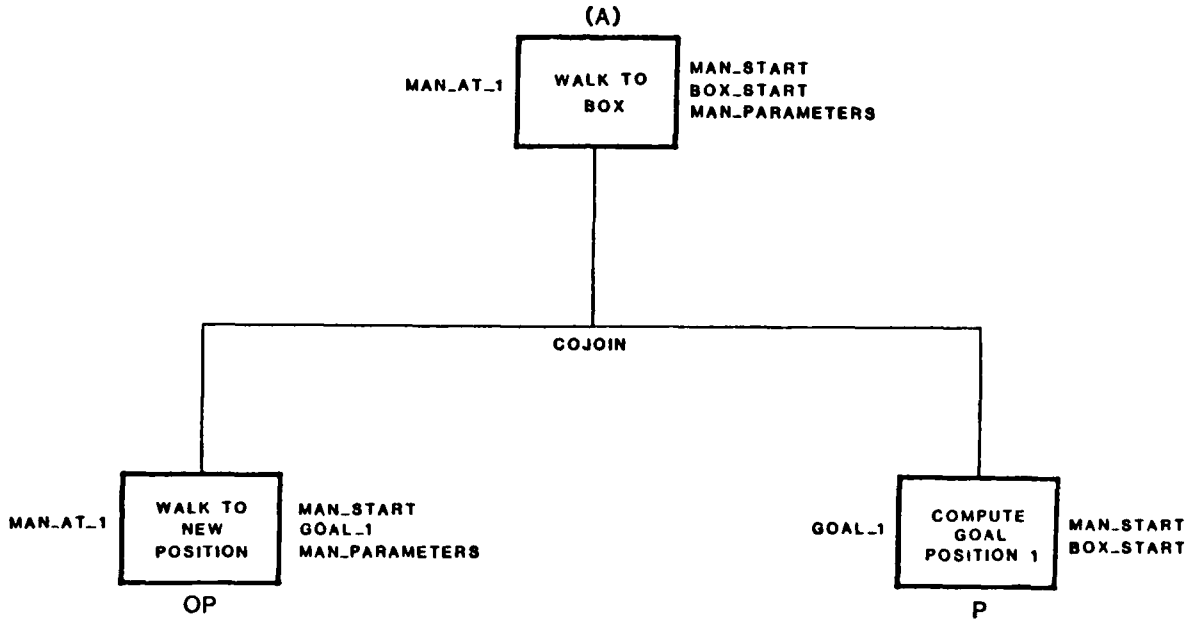


Figure 14B Control Structure for Example 2

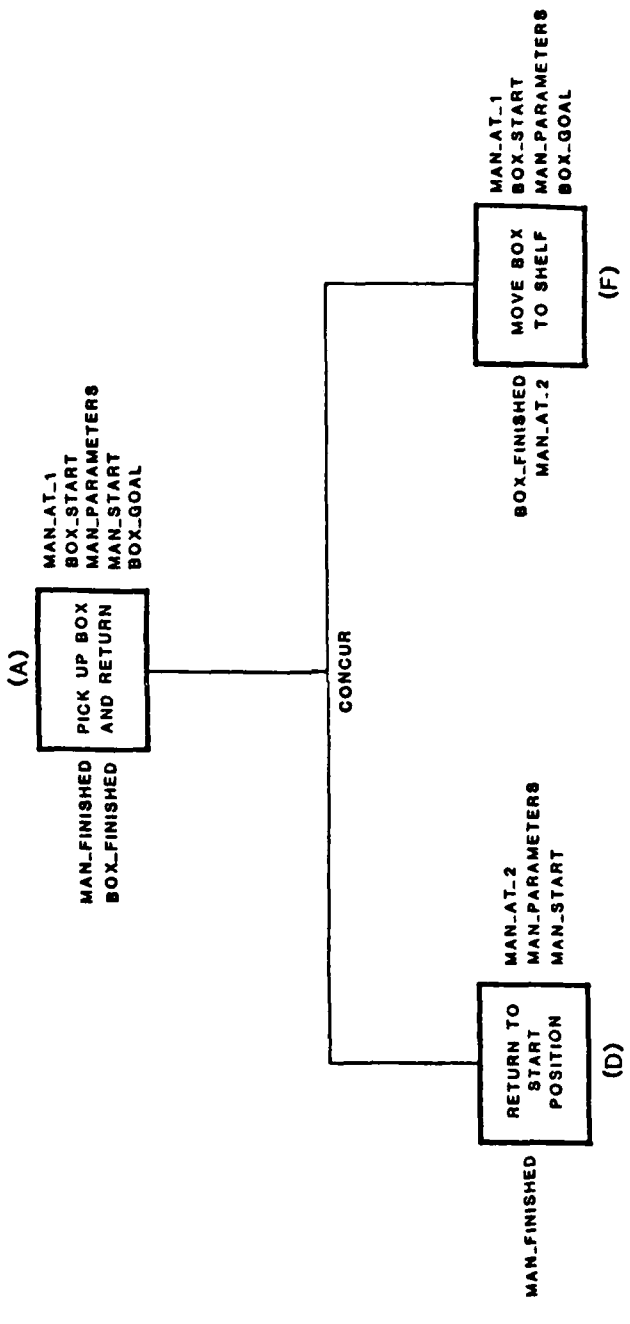


Figure 14C Control Structure for Example 2

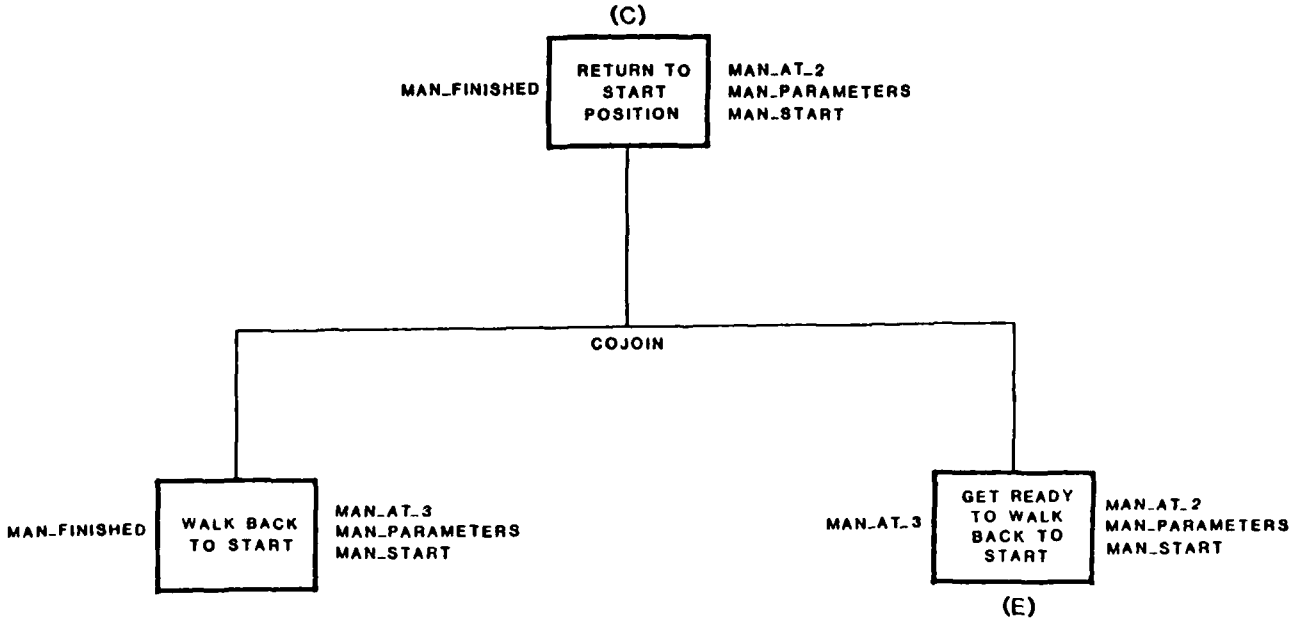


Figure 14D Control Structure for Example 2

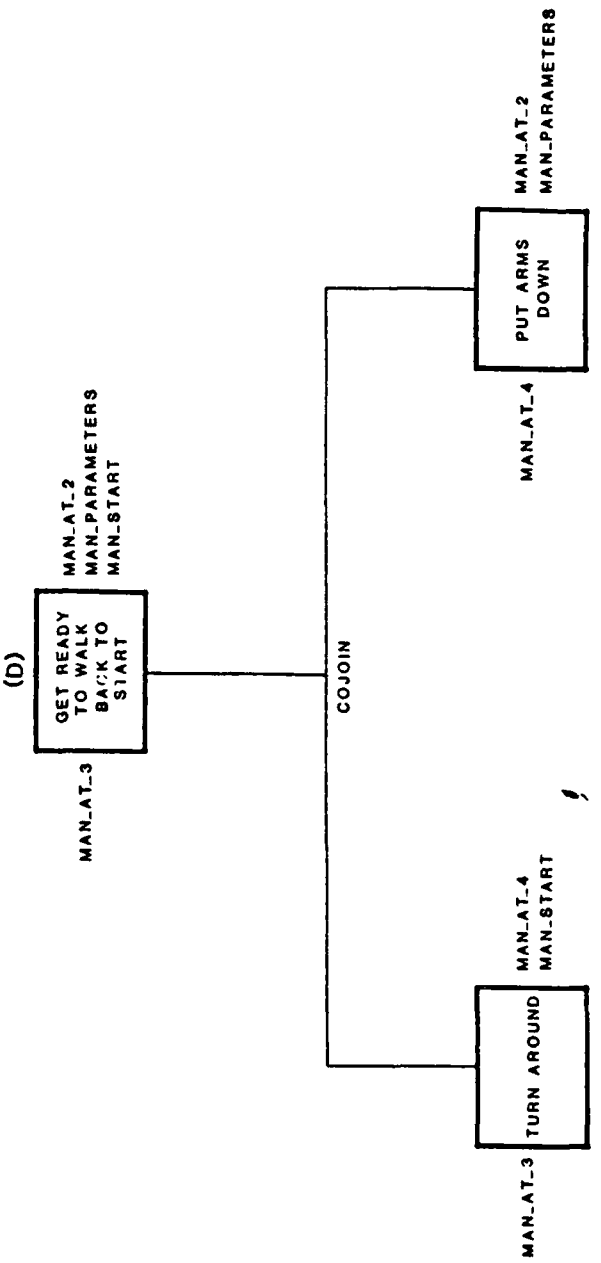


Figure 14E Control Structure for Example 2

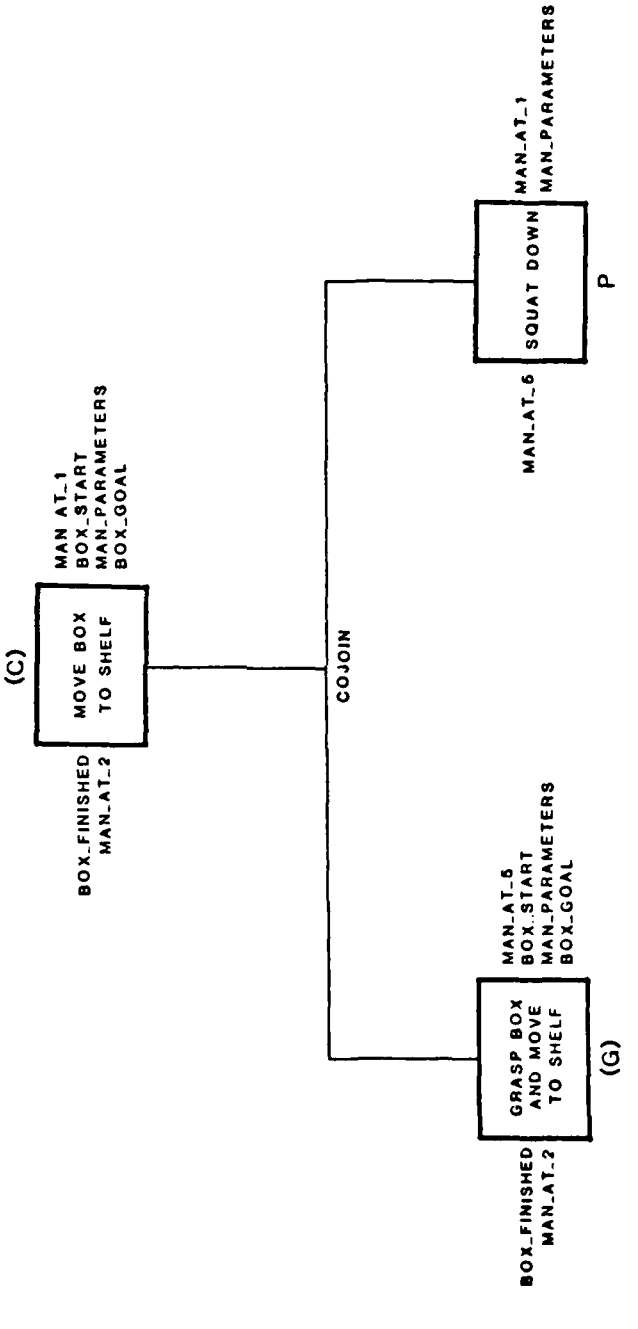


Figure 14F Control Structure for Example 2

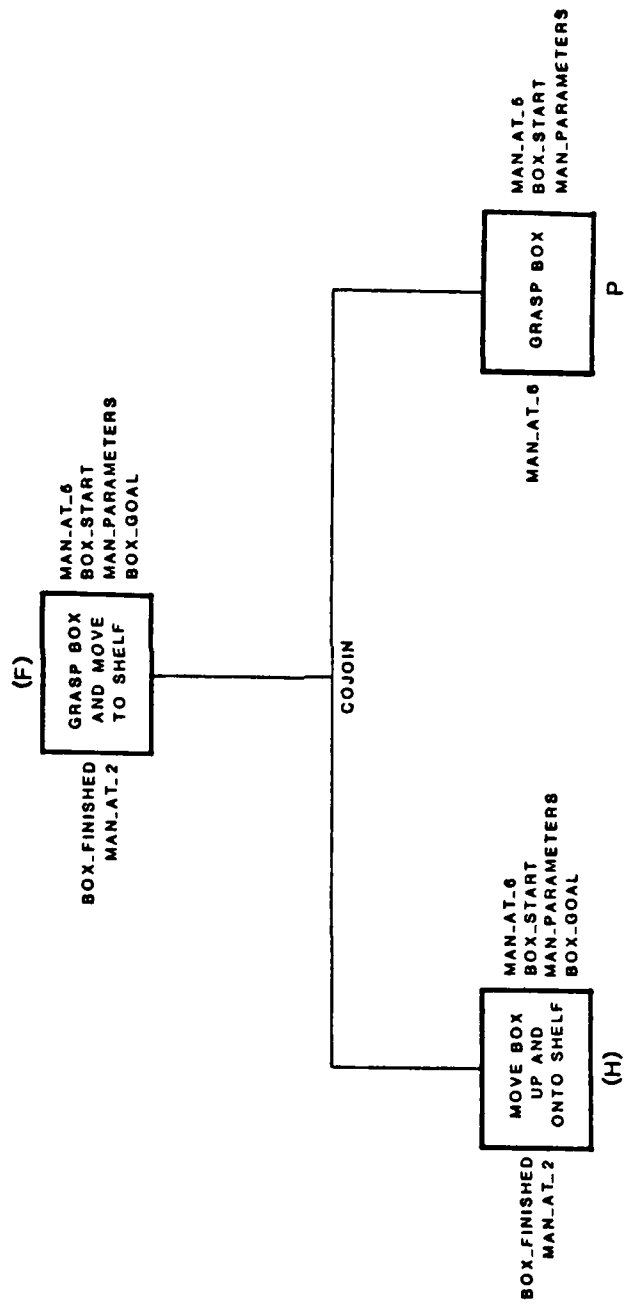


Figure 14G Control Structure for Example 2

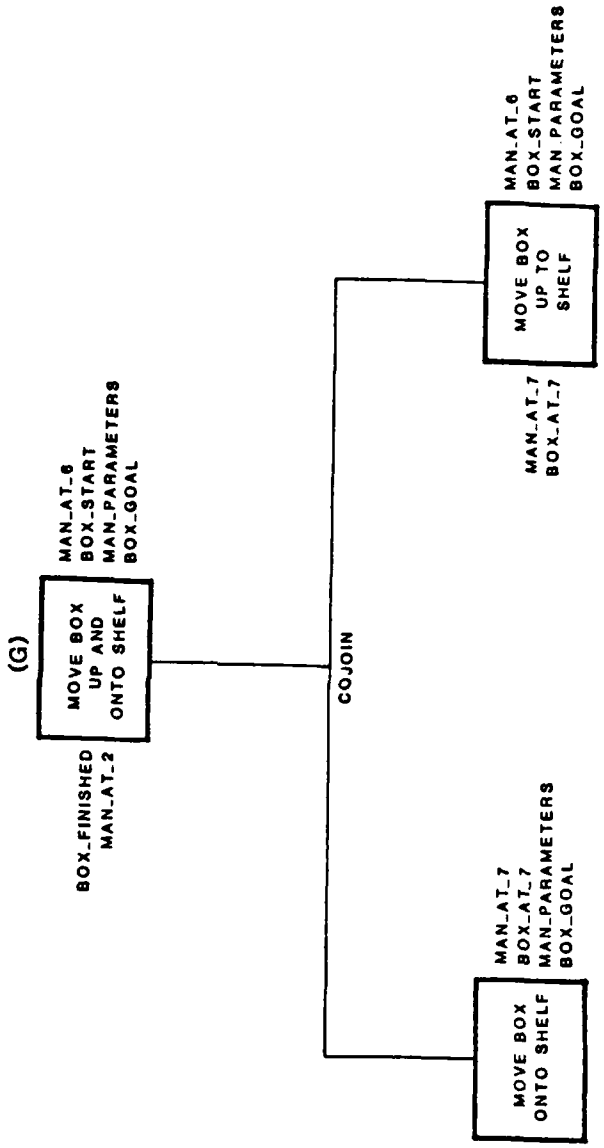


Figure 14H Control Structure for Example 2

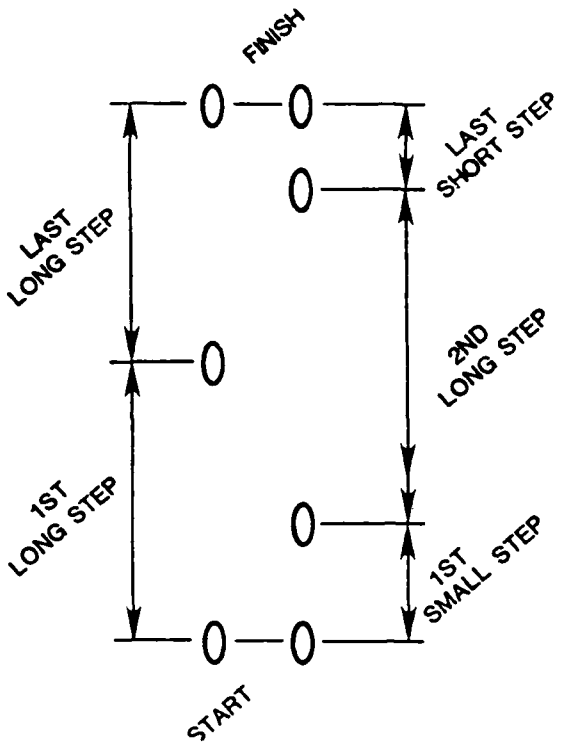


Figure 15 Step Pattern

All inputs/outputs of Figure 16 are not yet defined in detail. The development of a control structure can go a long way before the detailed definitions become necessary. By working first with the control structure before the detailed definitions, a significant part of the design can be accomplished at the abstract level of Figure 16.

MAN_FINISHED and BOX_FINISHED are Boolean variables to indicate when the box is moved and the man has returned back to the start position. MAN_PARAMETERS give the necessary details of the man. For example, it included his step lengths, height, arm parameters, etc. Nearly all remaining terms are geometrical information, referenced to the same yet undefined coordinate systems.

3.3 EXAMPLE 3, KEEPING A RADIO TUNED TO A SIGNAL

This example illustrates how human operator actions can be mixed with equipment functions within the same design context. A block within a control structure can represent equipment functions

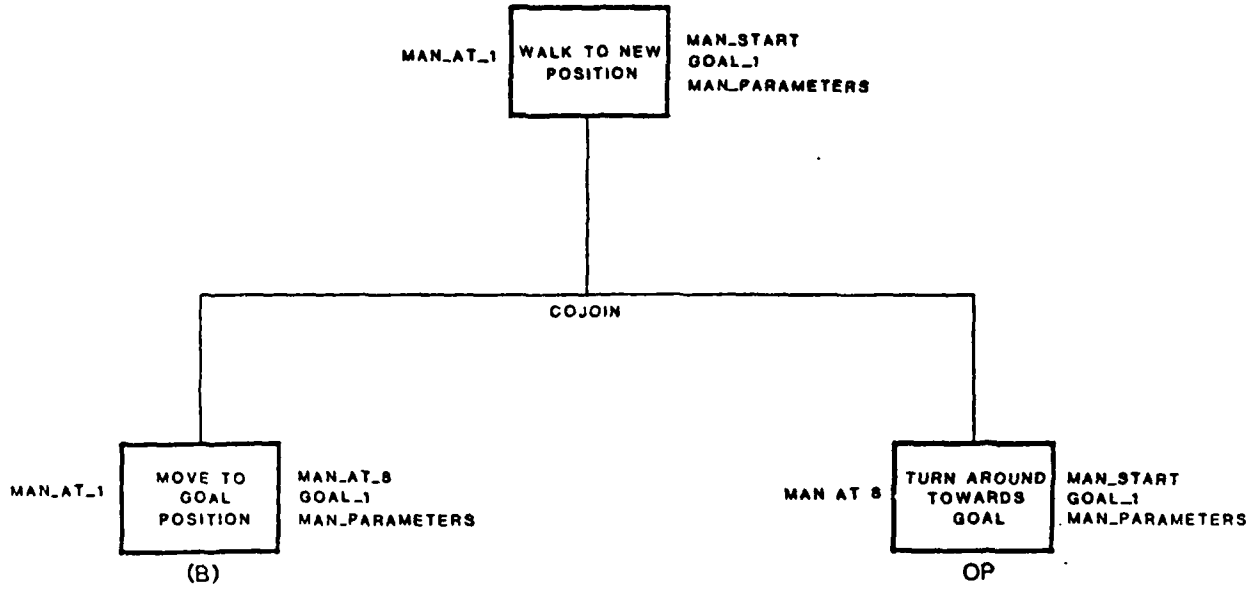


Figure 16A Control Structure

[redacted]

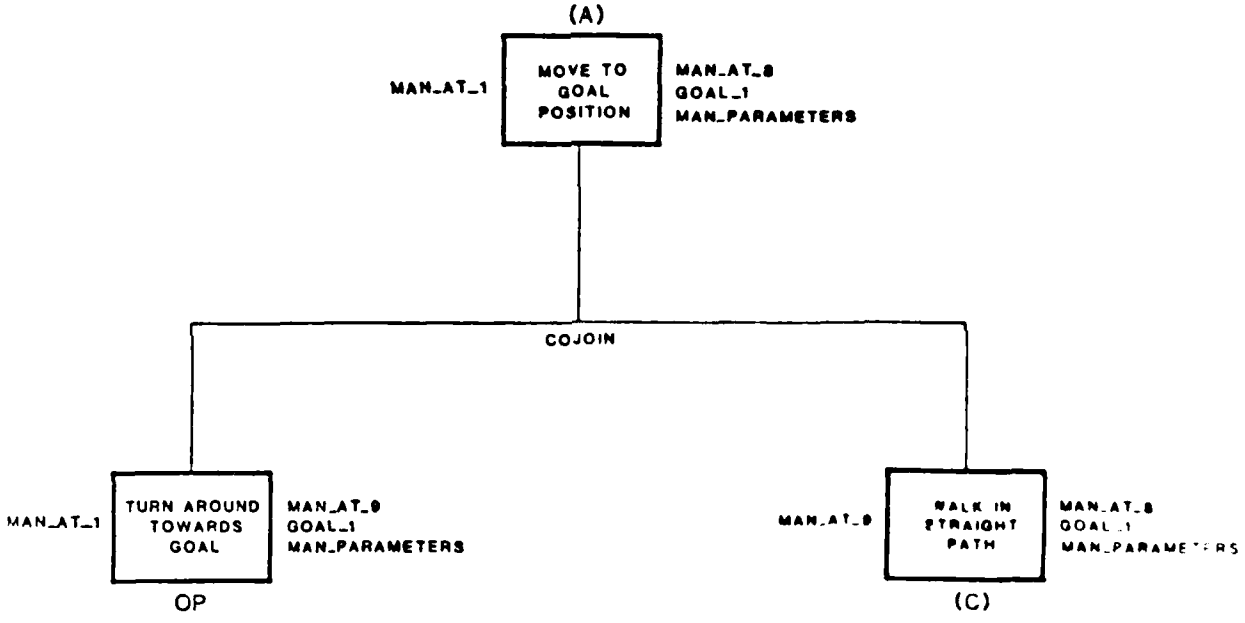


Figure 16B Control Structure

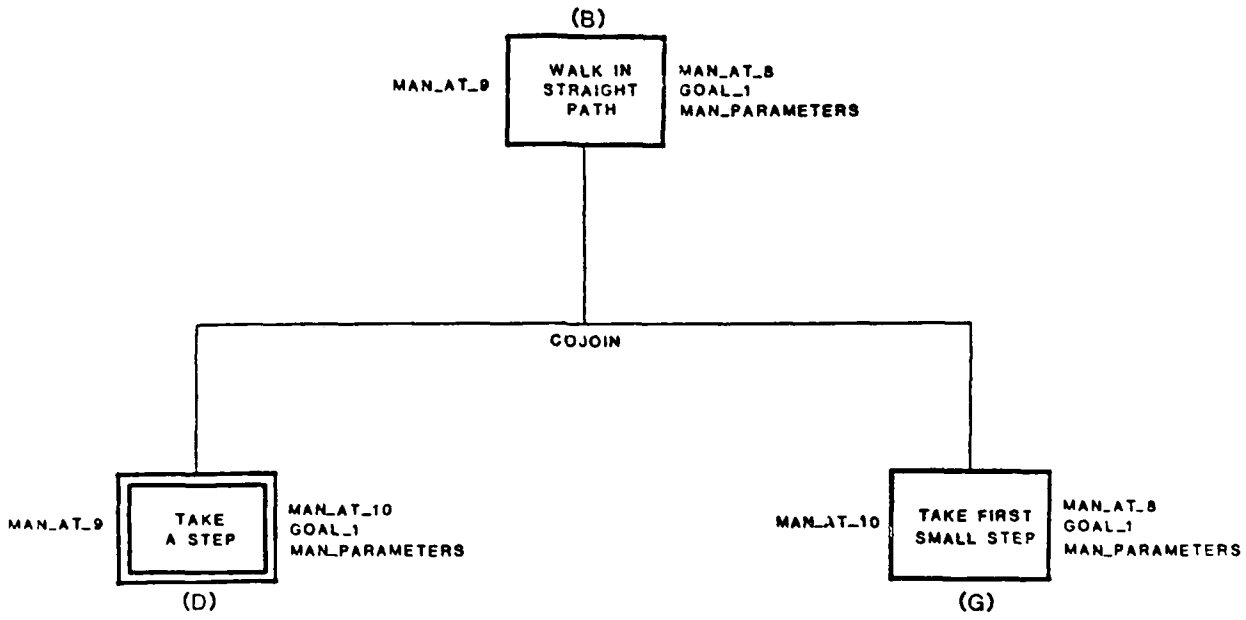


Figure 16C Control Structure

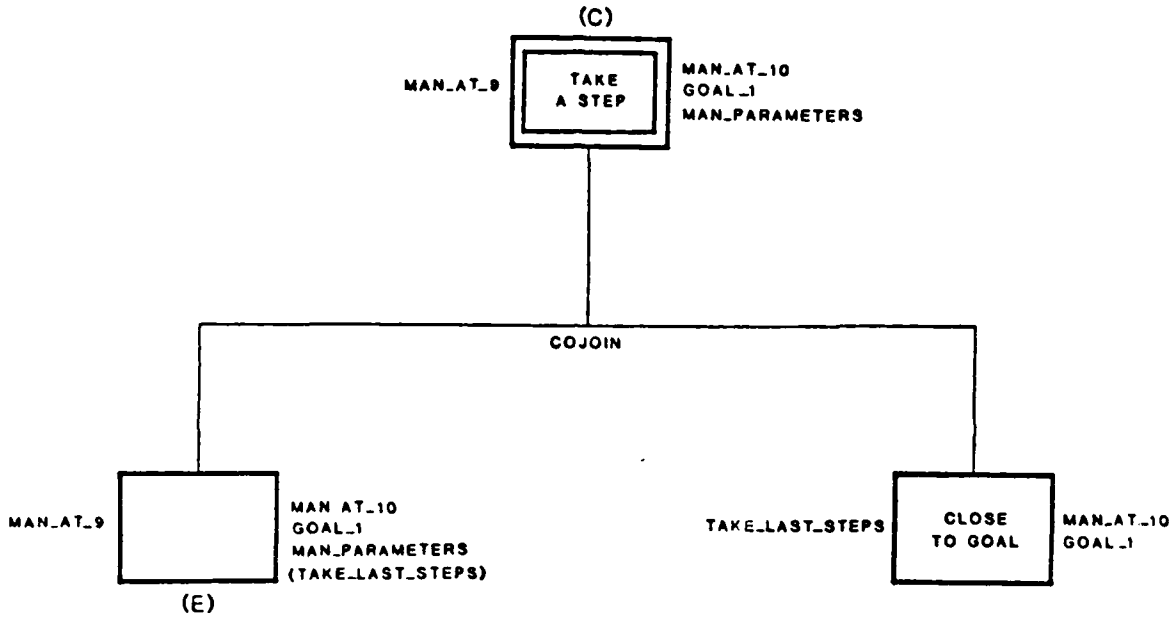


Figure 16D Control Structure

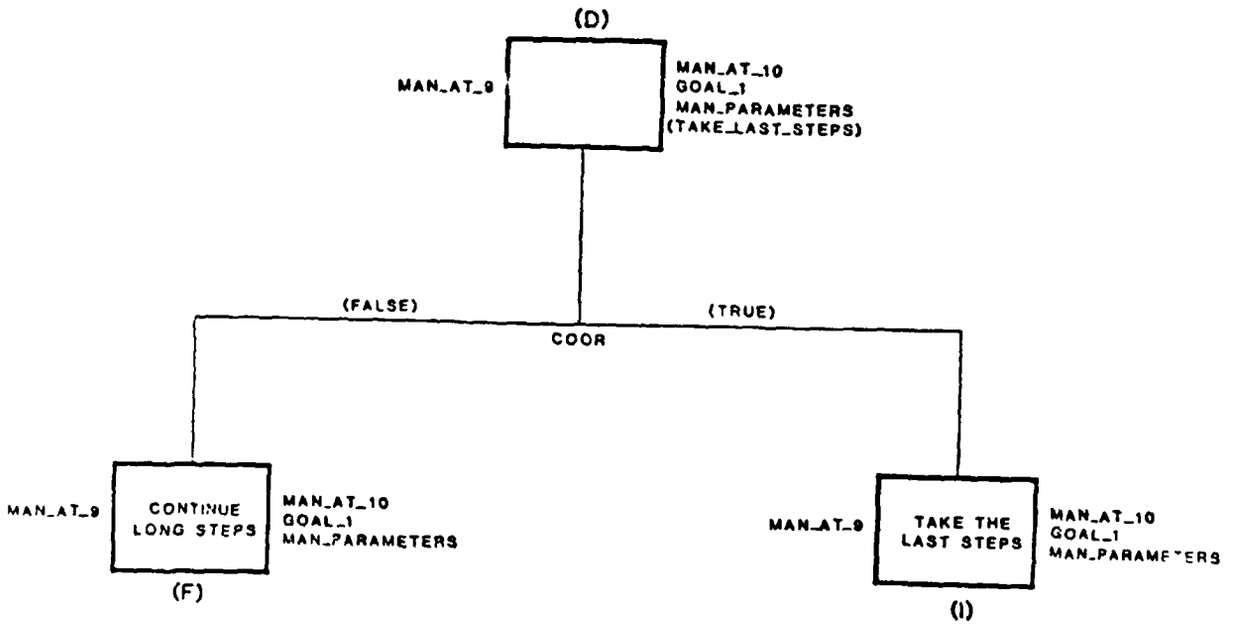


Figure 16E Control Structure

[redacted]

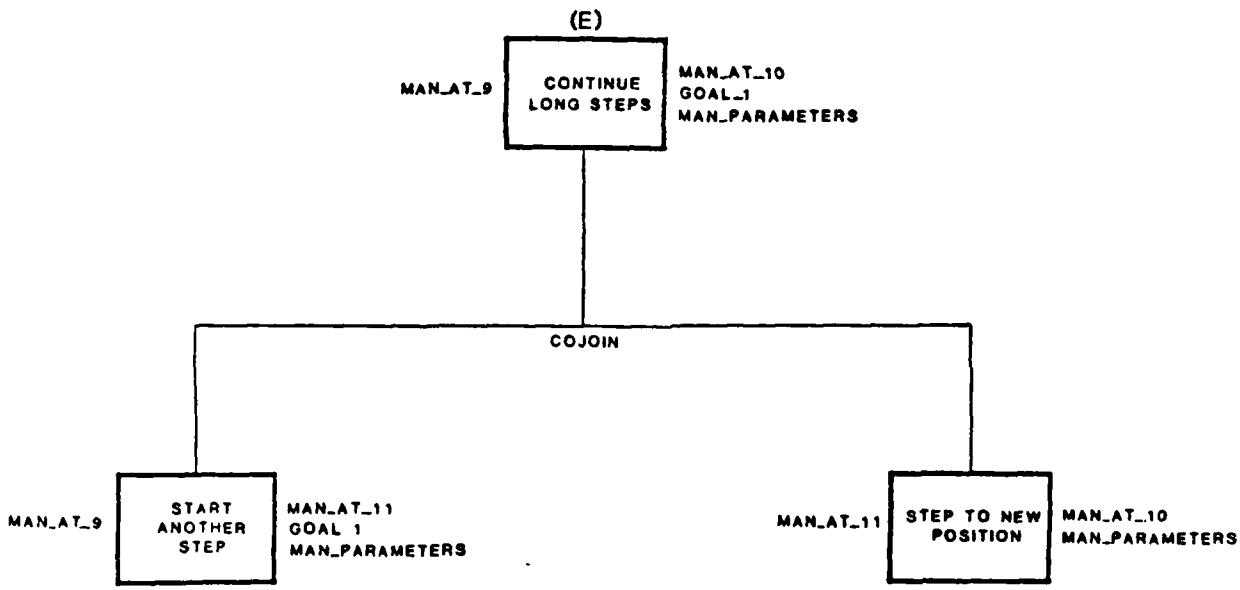


Figure 16F Control Structure

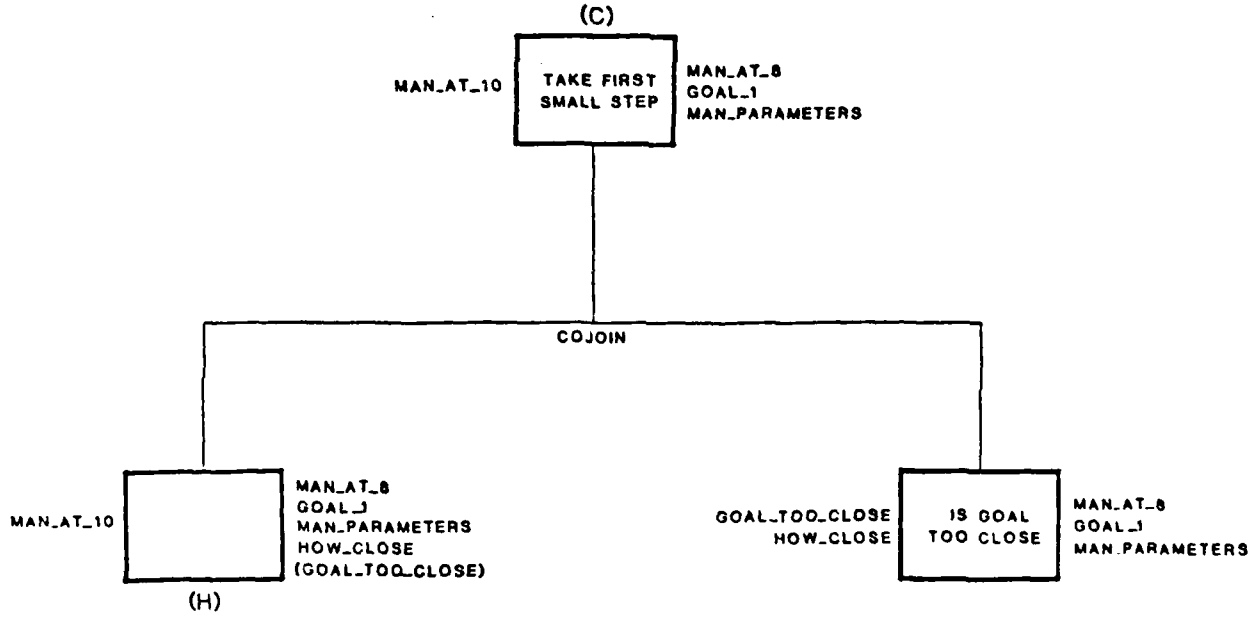


Figure 16G Control Structure

[redacted]

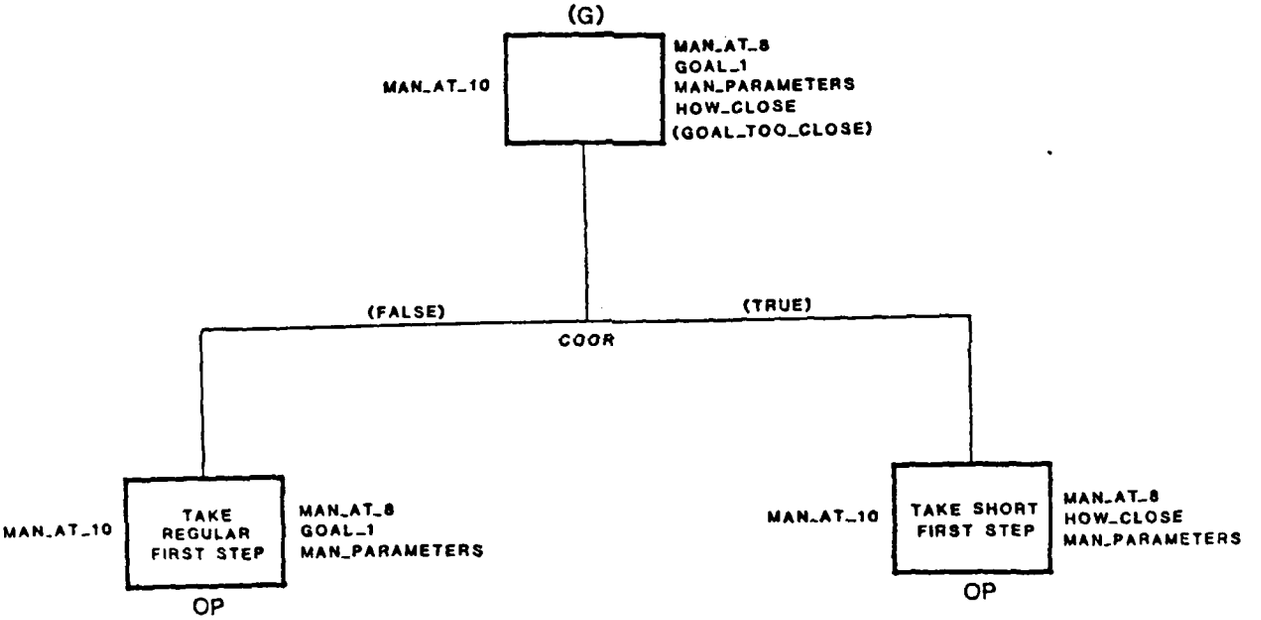


Figure 16H Control Structure

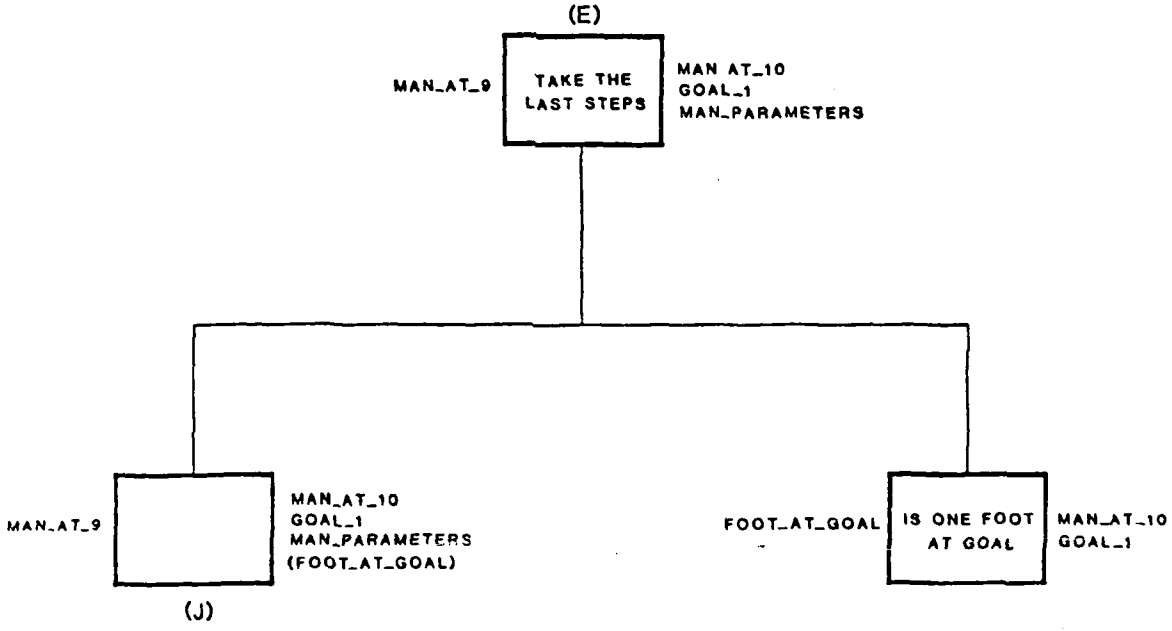


Figure 16I Control Structure

[redacted]

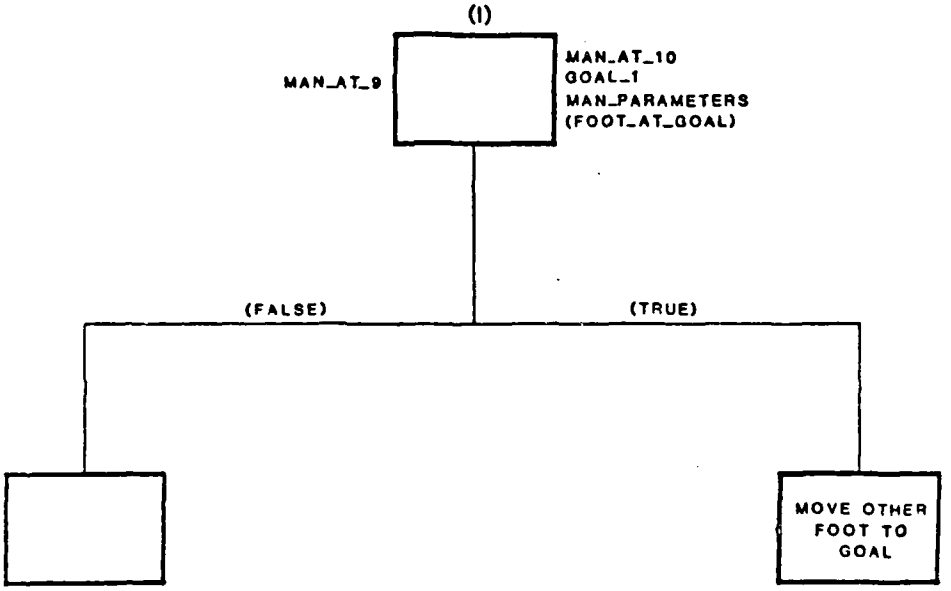


Figure 16J Control Structure

as well as any other function in which a set of inputs is transformed into a set of outputs. In this example, an operator listens to a radio. When the signal drifts, he reaches out to thetuning knob, turning it until the signal is clear again, returning his arm to its initial position. Figure 17 contains the control structure for this task.

The inputs and outputs at the root of the structure--the top process--are identical. This is appropriate for real-time applications in which previous outputs would be the inputs at the next increment of time. If the control structure is designed correctly, there will be no combination of conditions that will cause the control structure not to respond correctly.

3.4 EXAMPLE 4, A VARIATION ON EXAMPLE 3

One advantage of the USE.IT approach is that changes can be incorporated directly into the structure. This facility is important for human-factors applications since it will allow a basic system model to have psychological aspects added for test or evaluation purposes. For example, suppose that in Example 3, it was desirable to study the effects of hearing changes on the process of keeping a radio tuned. This effect would enter the control at the "determine signal quality" block of Figure 17A and "determine if signal satisfactory" block of Figure 17F. If a different radio were used, with a different process modeling the hardware, the "radio response" block of Figure 17N would only need to be changed. Another human factor variation might be to use a rate control instead of a positional control for tuning. That is, for a given level position, the rate of change of the tuned frequency would be constant. Conventional AM-FM tuning changes the frequency directly with the knob angle. "Grasp knob" of Figure 17C, "release knob" of Figure 17G and "start motion" of Figure 17E would have to be changed to accommodate a lever rather than a knob. The "rotate knob" structure beginning with the block in Figure 17J would have to be modified.

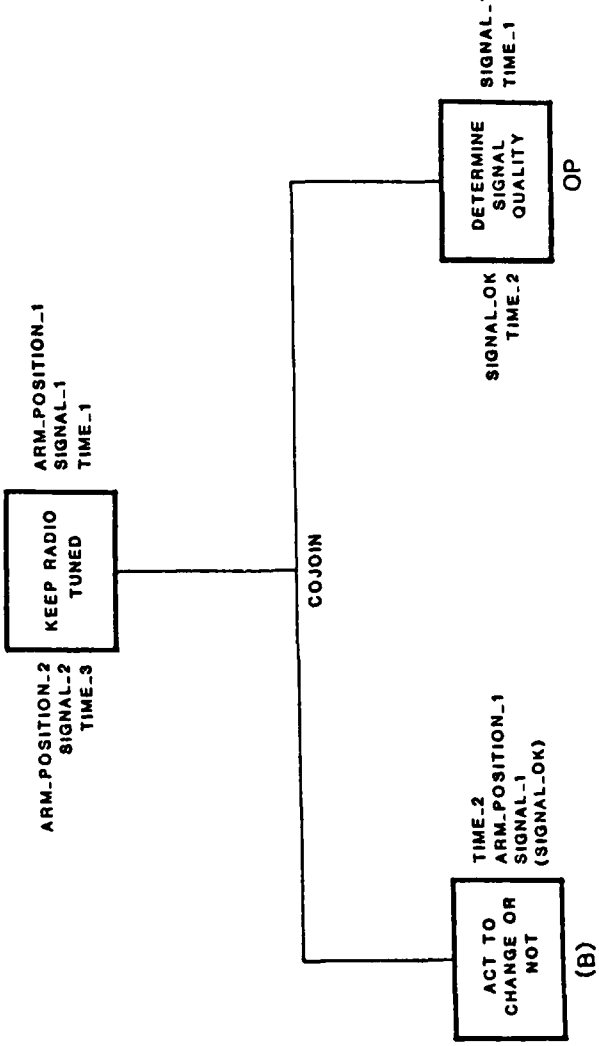


Figure 17A Control Structure for Keeping a Radio Tuned

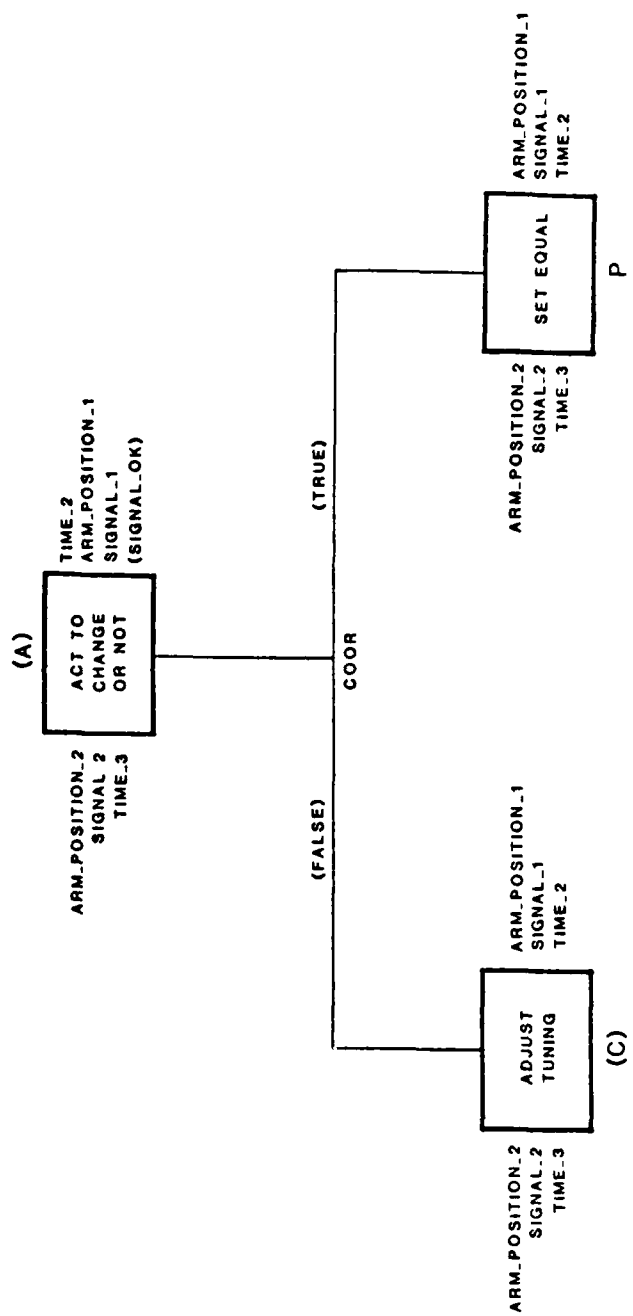


Figure 17B Control Structure for Keeping a Radio Tuned

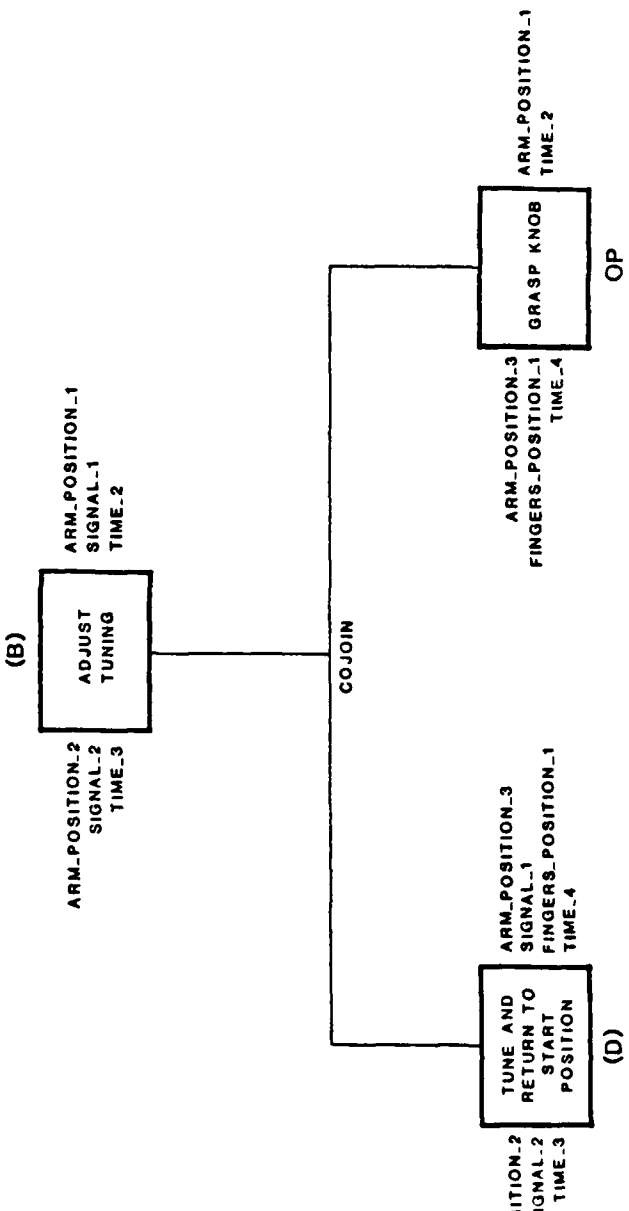


Figure 17C Control Structure for Keeping a Radio Tuned

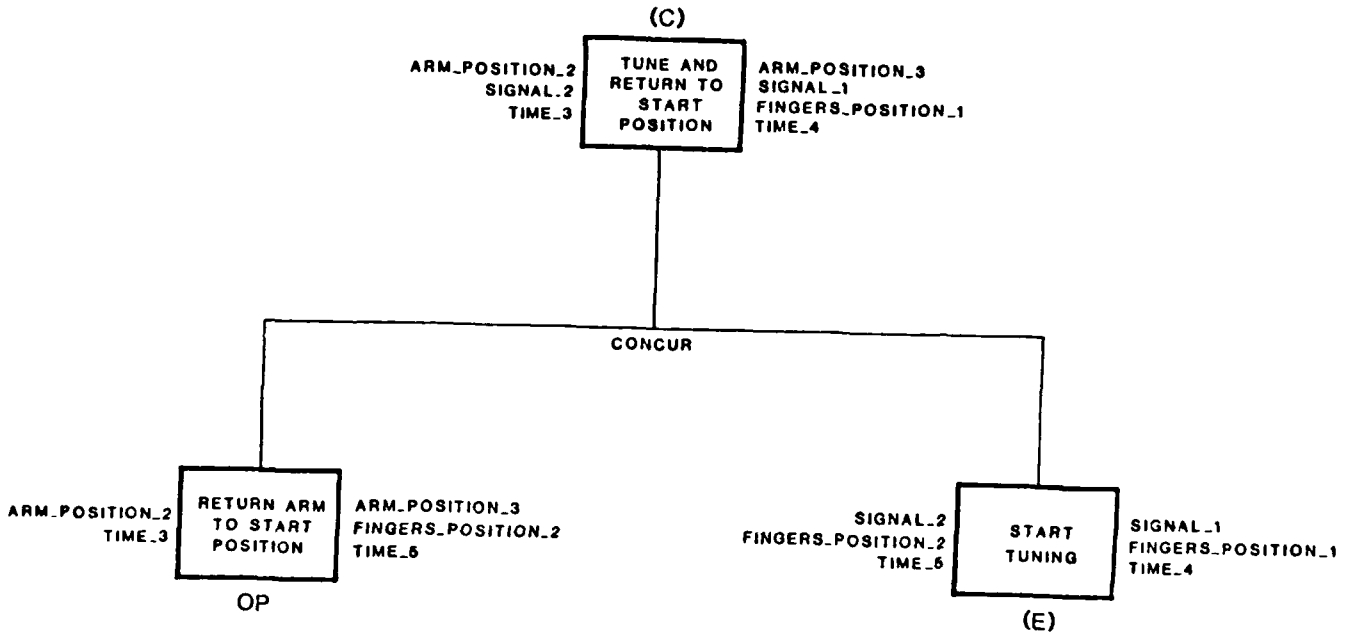


Figure 17D Control Structure for Keeping a Radio Tuned

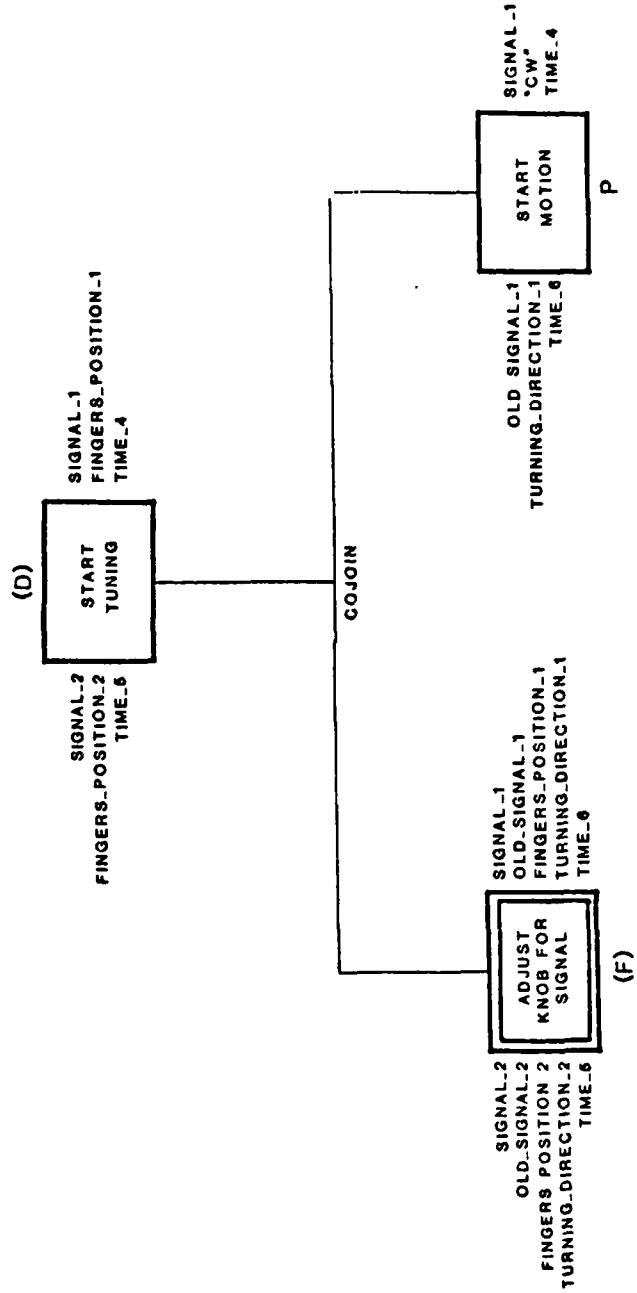


Figure 17E Control Structure for Keeping a Radio Tuned

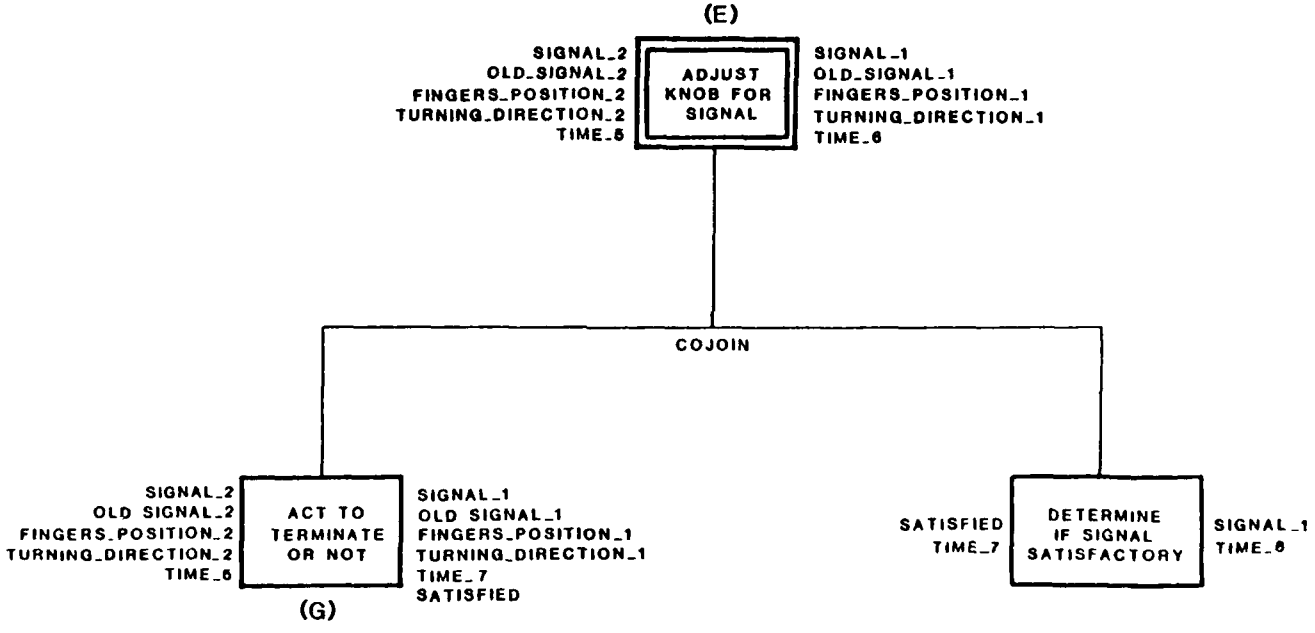


Figure 17F Control Structure for Keeping a Radio Tuned

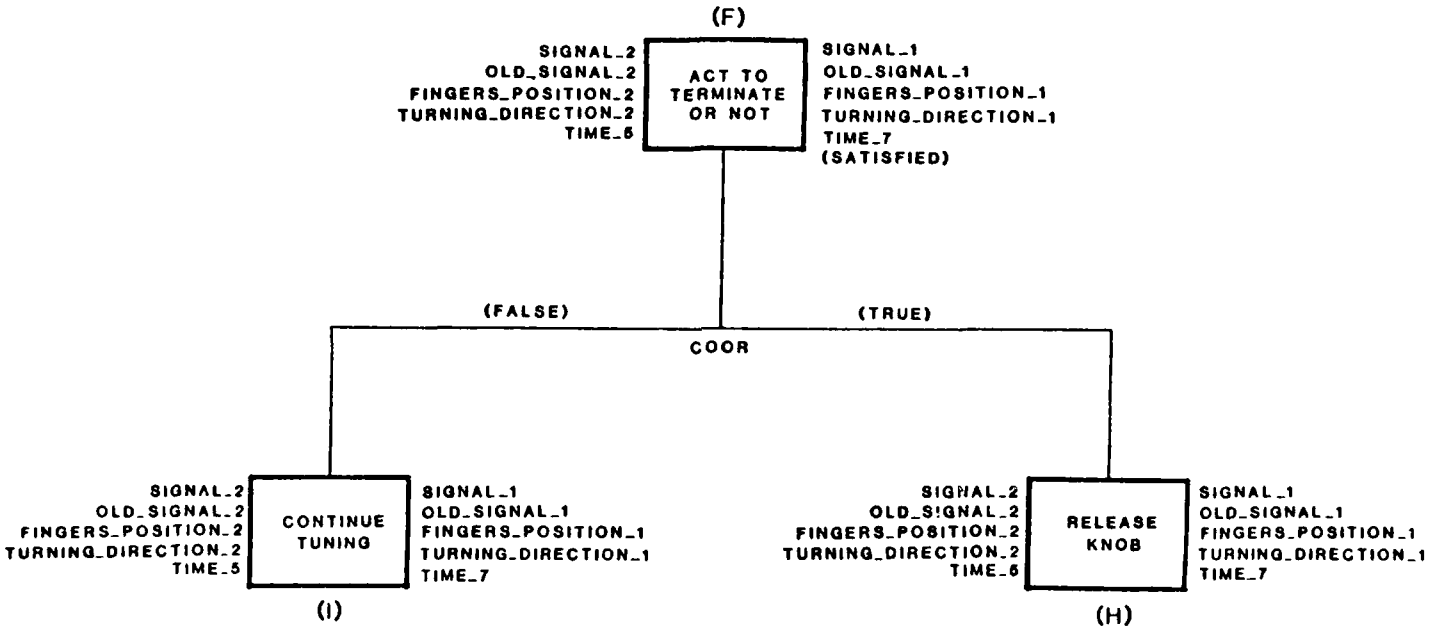


Figure 17G Control Structure for Keeping a Radio Tuned

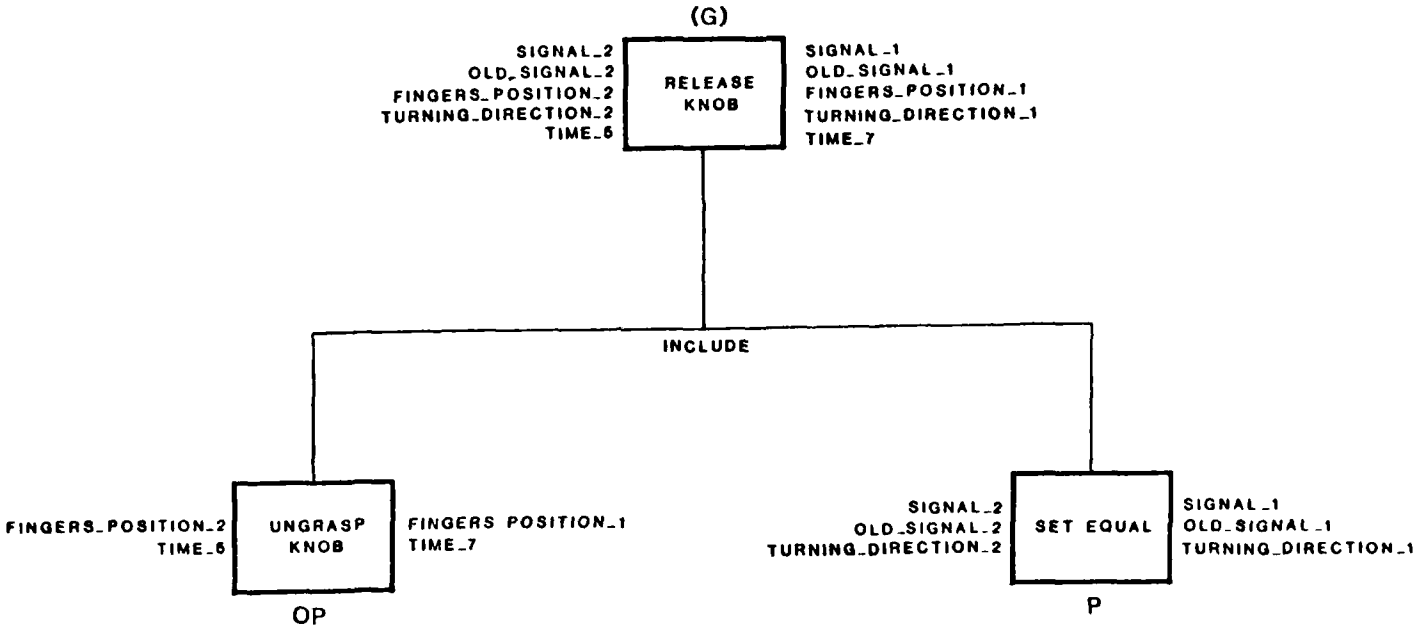


Figure 17H Control Structure for Keeping a Radio Tuned

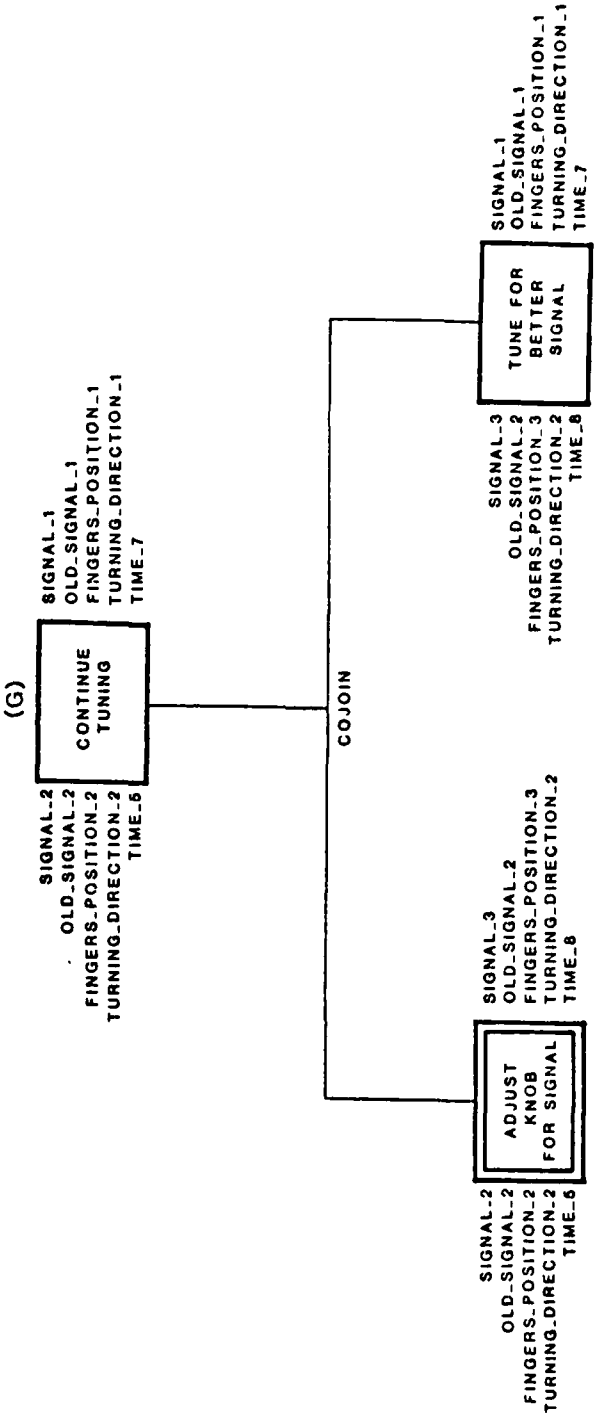


Figure 17I Control Structure for Keeping a Radio Tuned

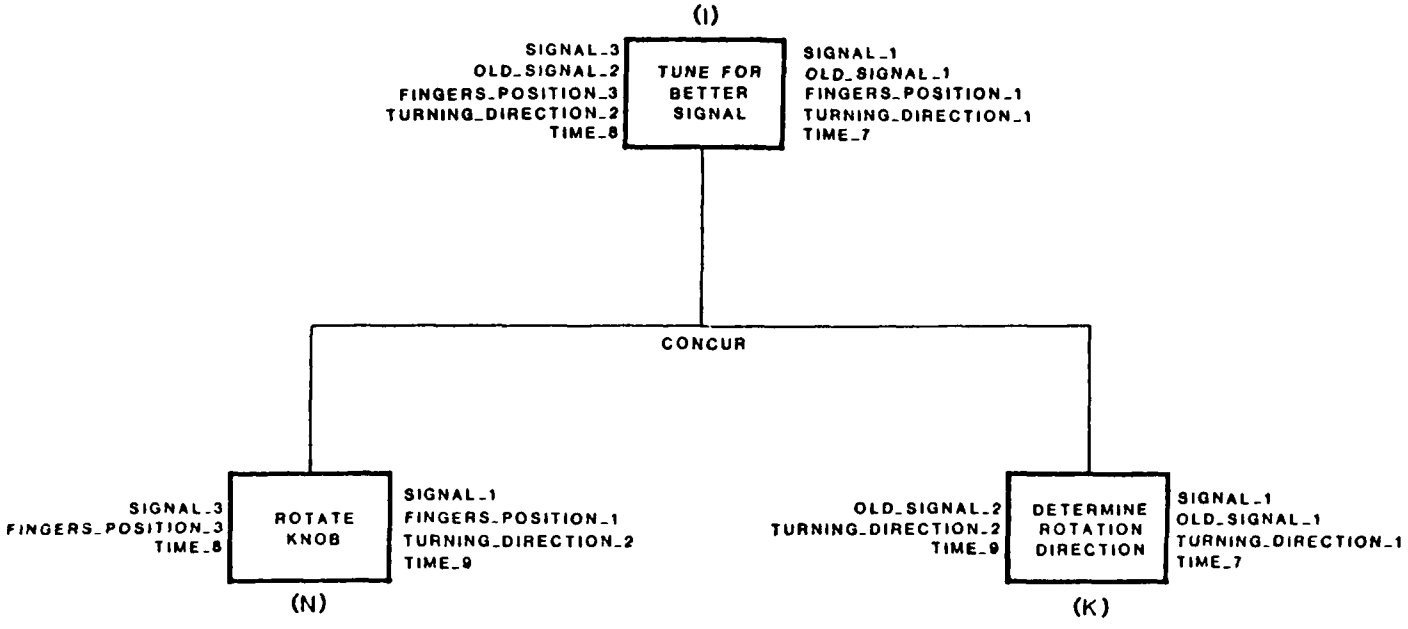


Figure 17J Control Structure for Keeping a Radio Tuned

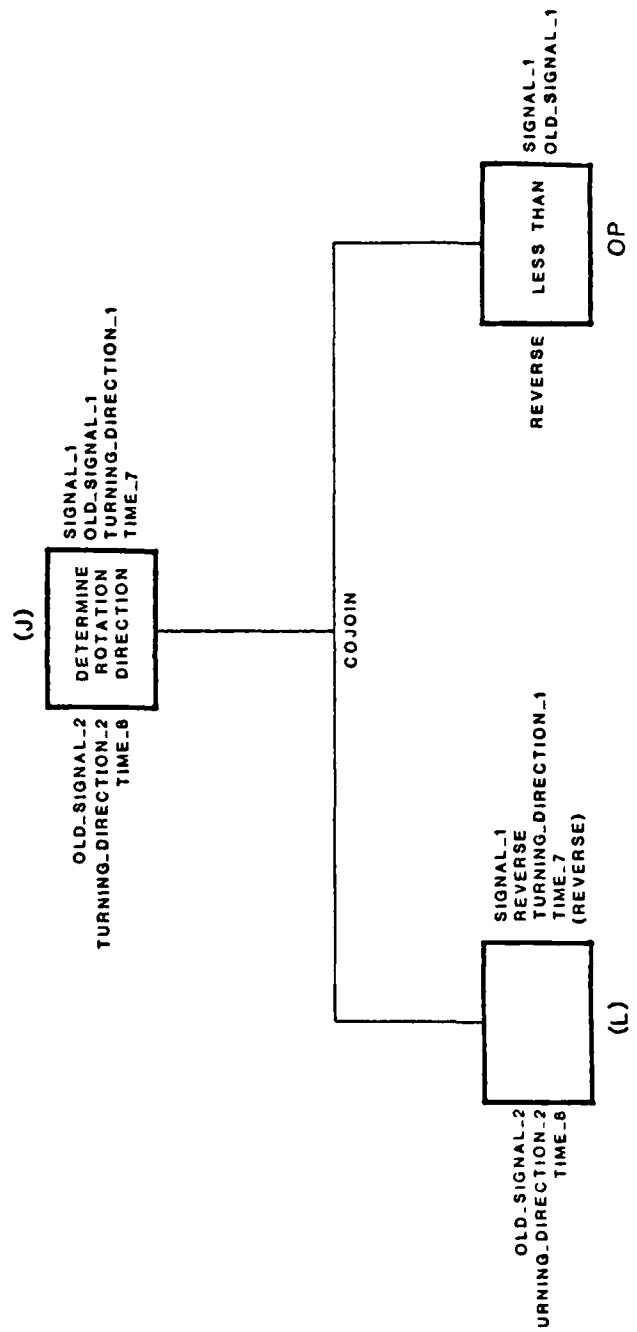


Figure 17K Control Structure for Keeping a Radio Tuned

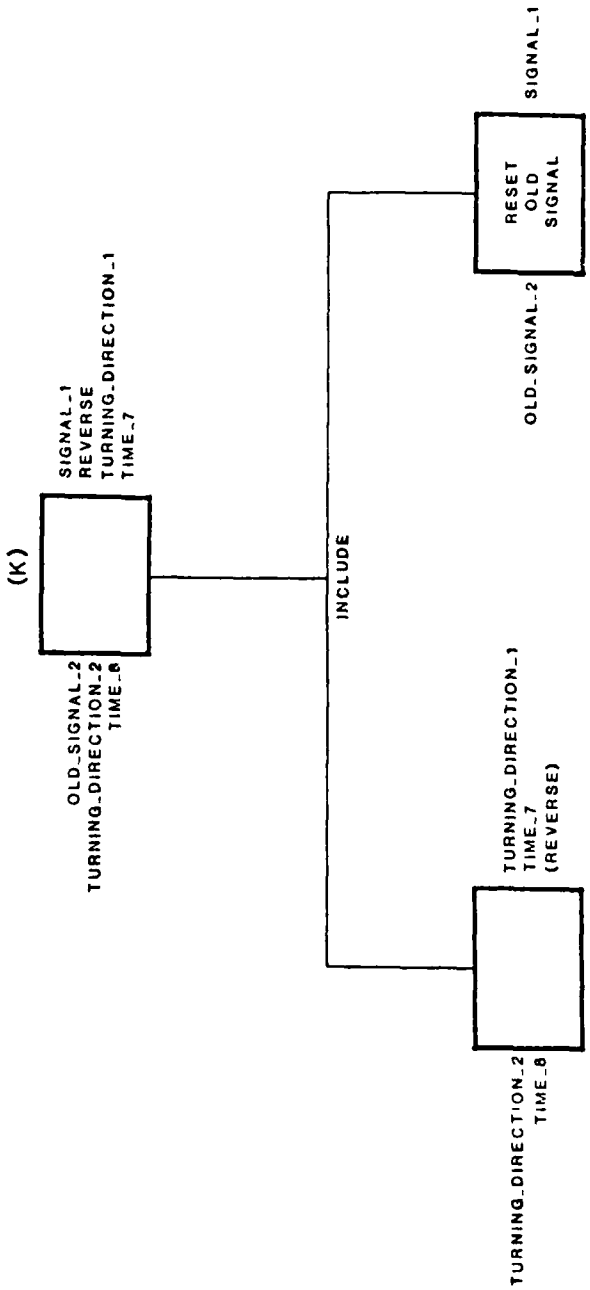


Figure 17L Control Structure for Keeping a Radio Tuned

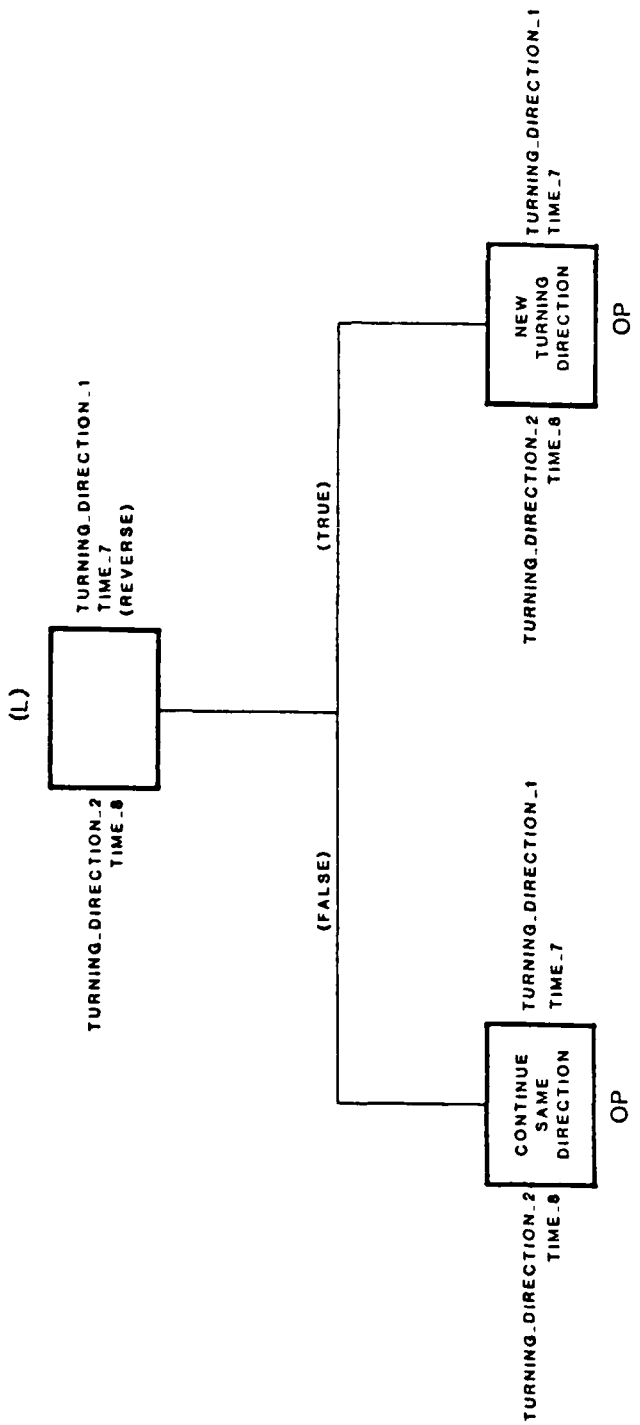


Figure 17M Control Structure for Keeping a Radio Tuned

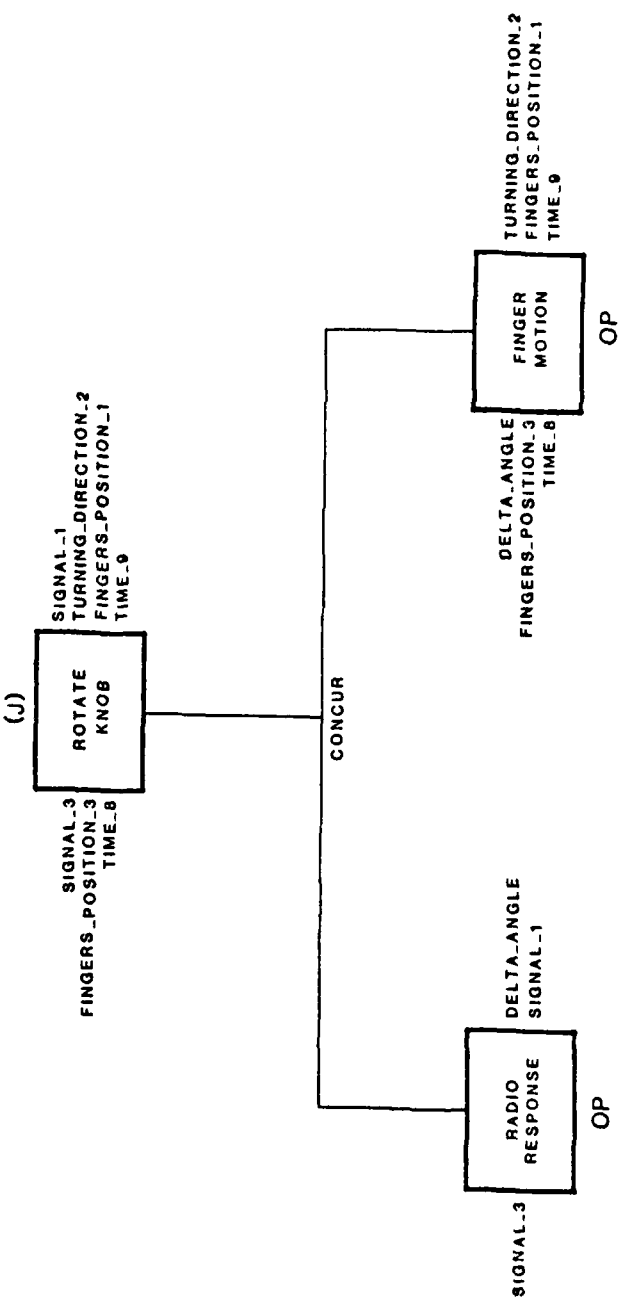
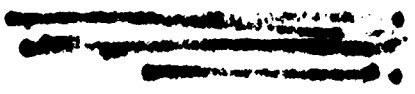


Figure 17N Control Structure for Keeping a Radio Tuned

[redacted]

Changes with variations for additional functions and levels of modeling always would involve changing part of the structure. With USE.IT, the exact areas to change can be easily identified. The control structure isolates functions with the inputs and outputs as specific interfaces.



4.0 DISCUSSION OF EXAMPLES

The application of a software design methodology to encompass all the details involved in complex man-machine interactions has been supported by the examples. This approach is based on the notion that the highly developed software design methods which model complex interactive procedures within software can be used to model complex interactive procedures between man and machine. The methodology can be extended without modification to interactions between machines, or between computer software and human operators.

The methodology assures that all effects, influences and dependencies are included within the modeling. Psychological processes, e.g., comparing oral signals to decide which way to turn a tuning knob are explicitly modeled as primitive processes, allowing psychological research to be directly related to a human factors applications. Assumptions about psychological phenomena are made explicit with this methodology, e.g., when a radio operator will quit trying to establish a conversation as shown in Figure 12F. Gaps in modeling become explicit with this method.

The analogy appears to be appropriate so that the USE.IT method can be usefully applied to human factors modeling.

4.1 STRENGTHS

The comprehensiveness of the methodology is demonstrated on the examples by USE.IT's application to establishing a radio linkage (Example 1), body kinematics (Example 2), and psychophysical tasks (Example 3). Equipment processes, e.g., radio functions, as well as purely psychological effects, e.g., human decision making have been included to demonstrate the breadth of the methodology.

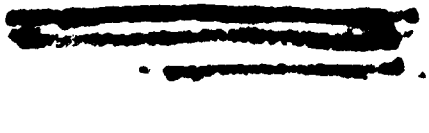
The many layers of abstraction, corresponding to the tiers of control structures correspond to the layers of software abstraction which have become a part of modern software

engineering. These layers of abstraction serve an important function within human factors modeling as well. For example, they allow extensions to be easily made to an initial design. If an initial design is completed but there is a later need to include some psychological model USE.IT facilitates this. If a control structure includes a block corresponding to a typist entering text into a computer, then a psychological model describing the probabilities of various typing errors can be added to the structure by a simple modification. In effect, the layers of abstractions of the control structure provide numerous places to add psychological or equipment extensions of the modeling.

The layers of abstraction add a dimension to the modeling which (1) allows the modeling to proceed hierarchically, (2) provides "hooks" on which to add model extensions, and (3) supports reusability of control structures.

Since each process, at each level, can be isolated and revised, this methodology encourages reuse of designs. Simple or complex structures can be accumulated within a library for reuse later. This approach to reuse of design efforts is formal and is consistent with the comprehensiveness of this modeling approach. Specifications can be formulated in terms of a process with inputs and outputs. The specification can be met by creating a control structure under this process which will yield the outputs given the inputs.

The USE.IT method makes the designer identify what factors are relevant in the modeling. For each level of abstraction of a control structure there is a clear distinction between what matters and what does not matter. This should make it easier to distribute components of the design task to specialists. For example, hardware engineers would have their area of modeling or design clearly defined. Human factorists would have their modeling clearly separated from the hardware modeling. This would be the case regardless of the complexity of the man-machine interfaces being modeled.



4.2 PROBLEM AREAS

The primary problem with the application of this methodology is its newness and its comprehensiveness. A new methodology which requires an absolutely complete attention to details is difficult to learn. Having to include all elements of a situation as explicit inputs or outputs is difficult.

Manually filling in structure chart forms, as in Figure 4 and 16, is tedious. Higher Order Software, Inc. has recently developed an automated version for PC computers which is relatively easy to use procedurally. However, whether manual or automatic, there will still be a conceptual difficulty in making a transition to this methodology. The power of the methodology is ready to be used, but it requires a large investment of training and study. This investment may not appear so large in light of the increasing complexity of man-machine systems. An automated, formal system is becoming more and more necessary, just like automated, formal design methodologies are almost necessary now for large software designs.

5.0 RECOMMENDATIONS FOR USE.IT'S APPLICATION

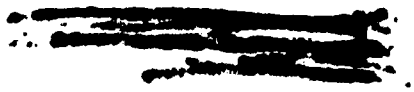
In the applied, practical aspects of doing human factors modeling, several guidelines have emerged. These guidelines will help anyone attempting this application of USE.IT for the first time.

The first guideline is to remain abstract as far down into the control structure as possible. Do not specify any detail before it is necessary. This "rational procrastination" approach encourages the designer to focus only on essential details. No time is wasted on irrelevant material.

A second guideline is not to worry about the excessive number of levels which are created in a control structure. If a level is irrelevant, its decomposition is impossible and it can be dropped. That is, it turns out that the second offspring block will not serve any function.

Another guideline is to create a new data term only when necessary within the control structure. If several terms overlap in the data that they contain, this may becomes obvious when they are both required as inputs at the top of the control structure. In other cases, the overlap will be obvious only when the data dictionary is elaborated. Some formal methods of data structure design would probably be helpful here. Examples are given in Reference 2 for data structure design.

As a guideline for real-time applications, the top block should have its set of inputs and outputs to be the same data. Of course, they will have different names, i.e., OLD_STATUS as input and a corresponding NEW_STATUS as output. If the USE.IT methodology is consistently applied, the relationship between the inputs and outputs at the root block at the top level will always be true over time. Further comments about real-time application can be found in Reference 3.



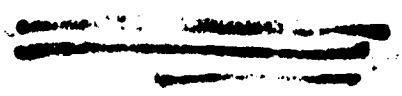
The best long-term application of this methodology will be the creation of model libraries of model components. As modeling of various systems is accomplished, reusable components can be kept within a library to be reused later. For example, the walking which was modeled in Example 2 can be applied to many other human-factors applications. This model reusability can be extended greatly by recognizing that parts of control structures can be taken out of their original, complete structure to be used in other models. The ability to isolate structure components and recombine them into new models maximizes reusability. Modeling with true reusable components can be achieved.

6.0 REFERENCES

(1) Harold P. Van Cott, and Robert G. Kinkade, (Eds) Human Engineering Guide to Equipment Design (Revised Edition) American Institutes for Research, Washington, D.C., 1972.

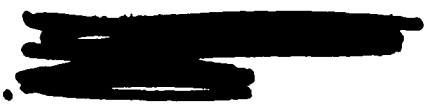
(2) James Martin, System Design from Provably Correct Constructs, Prentice-Hall, 1985

(3) Higher Order Software, Inc., The USE.IT System Reference Manual, VAX-Release 3.0, Higher Order Software, Inc. 35 Medford St., Somerville, MA, 02143, July, 1985.



APPENDIX A

Looping With USE.IT Structure



Within USE.IT, looping is treated as recursion. A control structure is created in which a block is defined with its structure including a reference to itself. By following a set of recursion rules, USE.IT can incorporate looping within its rigorous structure. More information on USE.IT looping is included in References 2 and 3.

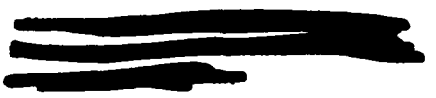
Figure A-1 shows a simple FORTRAN loop to compute the sum of the numbers from 1 to N. The control structure

```
      SUM = 0
      DO 10  I = 1, N
      SUM = SUM + I
   10 LOOP
```

Figure A-1 Simple Loop Program in FORTRAN

corresponding to this program is shown in Figure A-2. The two identical blocks for the recursion are indicated by double-lined borders. Level 0 is the root of the structure. Level 1 contains the COJOIN that initializes the looping and enters the loop function for the first time. Level 2 uses a Boolean variable FLAG to determine whether or not to exit the loop. FLAG is set to TRUE if index I is equal to N. The action to exit is not taken until level 3 with the COOR that either terminates or continues the looping. If continuing, the COJOIN at level 4 does the computations for the loop body and then calls the loop function again. The form of the loop illustrated is a DO-WHILE loop. This can thus handle cases of executing a loop zero times, if the WHILE-condition is not satisfied.

A general form for looping control structures is given in Figure A-3. Each block is identified in terms of its general function for the loop.



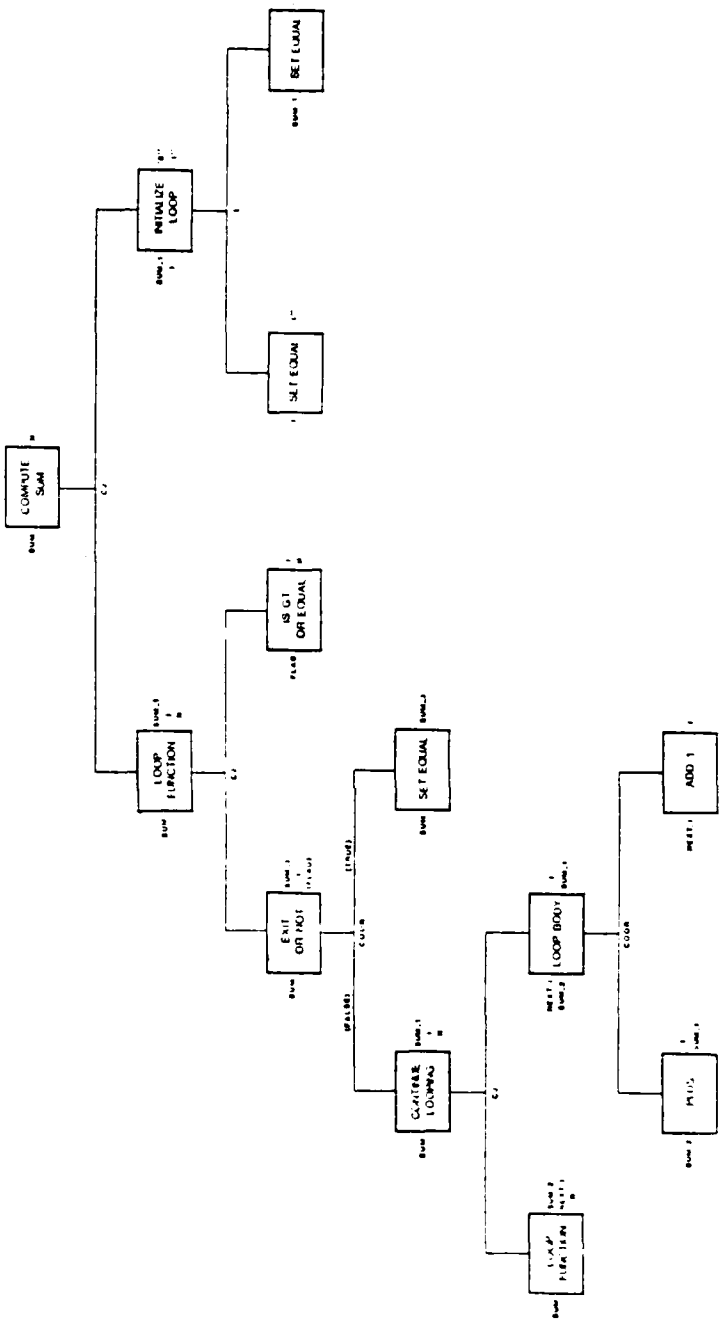


Figure A-2 Control Structure Corresponding To Do Loop

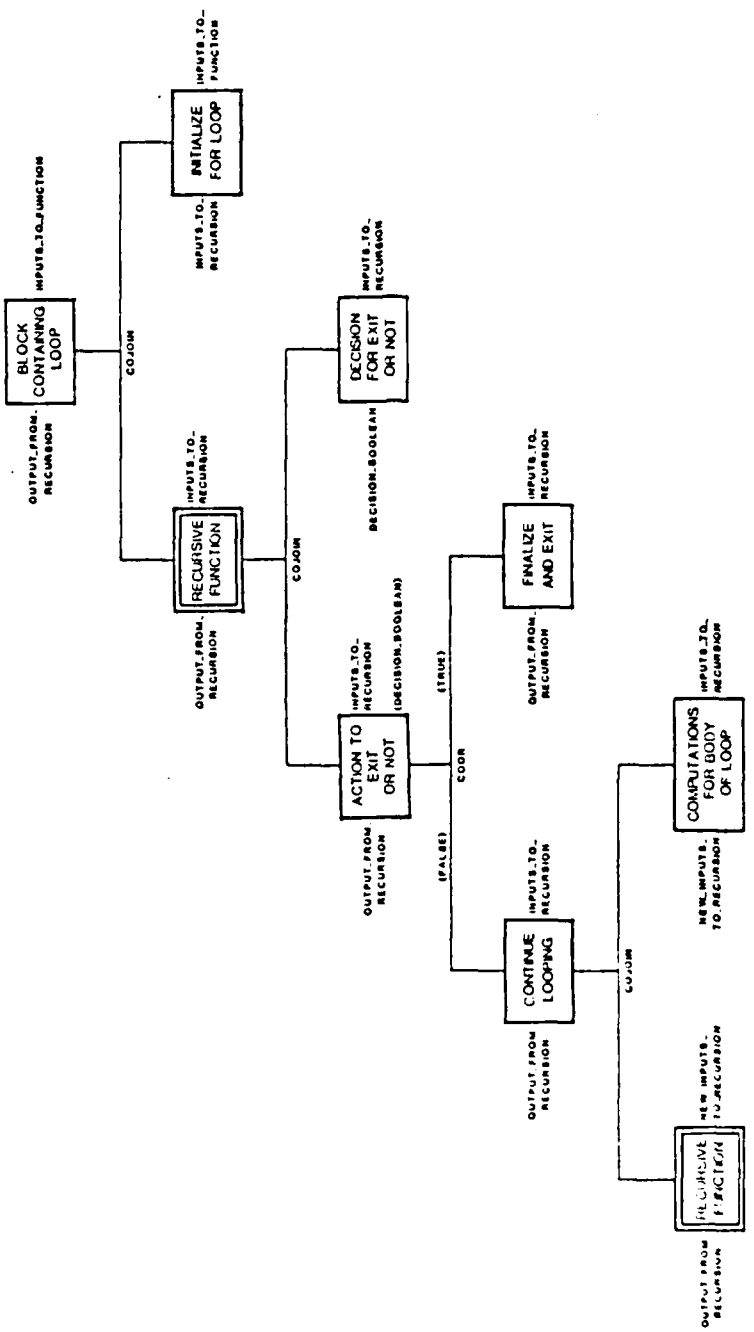


Figure A-3 A General Control Structure for Looping

END

DATE

FILMED

FEB.

1988